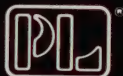




STD-LIB Library Reference





Pro-Log
Corporation

STD-DOS

Library Reference Version 2.1

**STD-LIB
LIBRARY REFERENCE
VERSION 2.1**

112606-003C 5/87

FOREWORD

The information in this document is provided for reference only. Pro-Log does not assume any liability arising out of the application or use of the information or products described herein.

This document may contain or reference information and products protected by copyrights or patents and does not convey any license under the patent rights of Pro-Log, nor the rights of others.

Printed in U.S.A. Copyright 1986 by Pro-Log Corporation, Monterey, CA 93940. All rights reserved.

Microsoft is a registered trademark of Microsoft Corporation.

MS is a trademark of Microsoft Corporation.

IBM is a registered trademark of International Business Machines Corporation.

Analog Devices is a trademark of Analog Devices, Incorporated.

CUSTOMER SUPPORT

For technical problems, please call Pro-Log Customer Support at (800) 538-9570, extension 3617. In California, call (408) 646-3617 from 7 a.m. to 4 p.m. (Pacific Time).

PRO-LOG CORPORATION SOFTWARE END-USER AGREEMENT

NOTICE: BREAKING THE SEAL ON THE DISK INDICATES YOUR ACCEPTANCE OF THE FOLLOWING TERMS AND CONDITIONS REGARDING THE USE OF THE COMPUTER PROGRAMS CONTAINED IN THIS PACKAGE. YOU SHOULD CAREFULLY READ THEM. IF YOU DO NOT AGREE WITH THE FOLLOWING TERMS AND CONDITIONS, YOU SHOULD PROMPTLY RETURN THE PACKAGE WITH THE DISK SEAL UNBROKEN AND YOUR MONEY WILL BE REFUNDED.

License

Pro-Log Corporation licenses the copyrighted computer software contained in this package, as follows:

1. Any application program you generate by linking to any of the runtime libraries (STDLIBCS.LIB, STDLIBCM.LIB, STDLIBCL.LIB, STDLIBB.LIB, STDLIBAL.LIB) or any of the object files with a name ending in .OBJ may be copied and distributed for use on systems having the associated STD BUS hardware products without payment of royalties. A copyright notice reading "PORTIONS COPYRIGHTED BY PRO-LOG CORPORATION, 1986" must be displayed on the media.
2. The STD-LIB source code or any source files with a name ending in .ASM supplied in this package may not be distributed without first entering into a license agreement with Pro-Log Corporation for such distribution.
3. All other software in your STD-LIB package cannot be duplicated except for the purpose of backing up your software. Other duplication of any of the software in the Pro-Log Corporation STD-LIB package is illegal.
4. You may transfer the program and license to another party if the other party agrees to accept the terms and conditions of this Agreement. If you transfer the program, you must at the same time either transfer all copies to the same party or destroy any copies not transferred, including all modifications and portions of the program contained or merged into other programs. Any such transfer automatically terminates your license to use the program.

Except as set forth above, you may not use, copy, modify or transfer the program, or any copy, modification or merged portion of it, in whole or in part.

From time to time, Pro-Log may advise you of corrections, enhancements, and updates to the computer program, which will be deemed to have been made upon the same terms and conditions as set forth in this Agreement.

Patents and Copyrights of Others

Pro-Log has no knowledge of any patent or copyright of others that is infringed by the computer program contained in this package. However, should there be made against you a legitimate claim by another of infringement of the other's United States patent rights or copyright by the computer program supplied under this order, Pro-Log will, at its own expense, defend any such claim and pay any award or damages assessed against you in any suit or proceeding based upon such a claim; provided, however, that Pro-Log is promptly notified by you of any such claim and is permitted to control any defense against the claim and its eventual resolution, and that you give Pro-Log all needed information, assistance and authority for Pro-Log to do so. In the event that your use of the

Pro-Log Corporation Software End-User Agreement (Continued)

program is enjoined by a court as the result of such a claim, Pro-Log will, at its option and within a reasonable time, either secure for you the right to continue using that product by suspension of the injunction, by procuring for you a license or otherwise, or to replace the product with a non-infringing one, or to remove the enjoined Product and refund the sums paid by you for the program. These provisions, however, shall not apply to a claim based upon a modified version of the program, in which case Pro-Log shall have no responsibility to you. The foregoing states Pro-Log's entire liability for infringement of copyrights or patents world-wide.

Limited Warranty

Pro-Log has made every reasonable effort to make sure that the program is accurate and free of errors but cannot guarantee this. However, Pro-Log does warrant for a period of one year from the date of initial delivery to you, that the disk media containing the machine code or the computer program is electrically functional and readable. Pro-Log's liability shall be limited to replacing any defective media which is returned F.O.B. Pro-Log's plant. Products which have been subject to abuse, misuse, alteration, neglect, unauthorized repair or installation are not covered by warranty. Pro-Log shall make the final determination as to the existence and cause of any alleged defect.

THIS WARRANTY IS IN LIEU OF ANY OTHER WARRANTY EXPRESSED OR IMPLIED. PRO-LOG MAKES NO OTHER WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING ANY WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE OR OF MERCHANTABILITY. IN NO EVENT SHALL PRO-LOG BE LIABLE FOR INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES SUCH AS LOSS OF PROFITS OR BUSINESS OPPORTUNITIES, NOR FOR THE INABILITY OF THE CUSTOMER TO USE THE MACHINE READABLE CODE.

General Terms

You may not sublicense, assign or transfer the license or the program, except as expressly provided above.

This license may be terminated by you by destroying the program together with all copies, modifications and merged portions. It will also automatically terminate if you fail to comply with any term or condition of this Agreement.

This Agreement shall be governed by the laws of the state of California. You agree that these terms and conditions are the complete and exclusive statement of the agreement between Pro-Log and you with respect to the computer program contained in this package, and that they supersede any proposal or prior agreement, oral or written, that relate to such a program.

CONTENTS

PAGE

Foreword/Customer Support	ii
Pro-Log Corporation Software End-User Agreement.....	iii
References.....	vii
Section 1. Introduction.....	1-1
Section 2. Operation and Programming	2-1
Section 3. Creating New Library Routines.....	3-1
Section 4. C Language Library.....	4-1
Section 5. BASIC Language Library	5-1
Section 6. Assembly Language Library.....	6-1

ACKNOWLEDGEMENTS

Program Design:	John Gill, Jim Gibbons
Programmers:	Dave Cannon, Jon Doelman, David Fahrner, Bill Haught Jennie Wei
User's Manual:	Jon Doelman, Carol Wall, Jennie Wei
Production:	Robert Fletcher, Bobbie Hall, Brad Hicks, Patricia Saulsbury
Editing:	Robert Fletcher, Bev Hughes
Word Processing:	Bev Hughes
Project Supervision:	Jim Gibbons, Carol Wall
Product Management:	John Orr

REFERENCES

Microsoft C Compiler 3.0 For MS-DOS Personal Computers.
Microsoft Corporation

Microsoft BASIC Compiler for the MS-DOS Operating System.
Microsoft Corporation

Microsoft Macro Assembler 4.0 For Personal Computers Running the MS-DOS Operating System. Microsoft Corporation

STD-DOS User's Guide. Pro-Log Corporation

C Programming Guide 2nd Edition, by Jack Purdum, Que Corporation, Indianapolis, 1983

Learning IBM BASIC, by David A. Lien, Compusoft Publishing, San Diego, 1984

The IBM BASIC Handbook, by David A. Lien, IBID (This book is a companion to the preceding book)

Assembly Language Primer for the IBM PC & XT, by Robert Lafore, The Waite Group, New American Library, N.Y., N.Y., 1984

Assembly Language Programming for the IBM Personal Computer, by David J. Bradley, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1984

Programmer's Guide to the IBM PC, by Peter Norton, Microsoft Press, Bellevue, Washington, 1985



Section 1

INTRODUCTION

CONTENTS	PAGE
Overview	1-1
System Requirements.....	1-2
System Components.....	1-3
Warranty Registration Card.....	1-4
How to Use This Manual	1-4
Getting Starting.....	1-5
Tutorial.....	1-7
Functional Description	1-15

INTRODUCTION

OVERVIEW

STD-LIB is a collection of software routines (object modules) for STD BUS I/O cards. The modules interface the real world of STD I/O with industry-standard operating systems and languages. The modules are designed to allow rapid application program development and debug in an STD-DOS environment.

Application programs may be written in any one of three different languages: Microsoft C, Compiled BASIC, or Macro Assembler. Any application program containing library object modules may be distributed for use on STD BUS systems without payment of royalties.

All source code used to create the library modules is provided on disk. The source code is as clear and understandable as is practical while maintaining efficiency. This allows you to modify functions to create new routines or to replace existing routines.

A specification for the library is included in Section 3, Creating New Library Routines. This section allows you to write routines for STD cards not currently supported in the library. New routines are easily developed by using the STD-LIB Include file and macros. These routines are written in assembly language, and the library software automatically provides for overhead tasks such as: interfacing to the different application languages, allocating scratch RAM, and saving register contents.

The library modules are maintained and organized under the Microsoft Library Manager (LIB). The library modules actually used by your application program are linked with your program by the Microsoft Linker (LINK).

SYSTEM REQUIREMENTS

During application program development, STD-LIB has the same hardware requirements as STD-DOS, which is used during development. STD-DOS requires an IBM PC, PC/XT, PC/AT or hardware compatible. The PC should have at least one 5-1/4-inch disk drive, 256K of system RAM, and one RS-232-C serial port.

The PC should use PC-DOS 3.0 or higher to be fully compatible with the STD-DOS utilities "STD" and "ROMDISK." The IBM PC/AT requires the addition of an adapter to connect the 25-pin RC702-3 (STD-DOS RS-232-C serial cable) to the 9-pin connector on the asynchronous communication port. The IBM part number for this adapter is 6450242.

After application program development is completed and the application program resides in ROM, only an STD-DOS system with sufficient memory to contain and run the program is required.

Development Software

Your application program must be written using Microsoft Macro Assembler, Microsoft BASIC Compiler, or the Microsoft C Compiler. These software packages are available from Pro-Log.

Optional Components

A **text editor** capable of producing ASCII format files with no control codes (such as the BRIEF editor by Underware that is available from Pro-Log) can make application program development of long programs much easier. Many text editors that normally use control codes or special formats for documents also provide a non-document or programming mode for producing ASCII files. However, a line editor like EDLIN, which comes with the PC, can also be used. Line editors work best for short programs.

A **PROM programmer** for placing the application software into PROM. The Pro-Log SYS 92-321 Gang Programmer and GC88 configurator program from one to eight 27256 PROMs simultaneously. Other configurators are available for the 8K-, 16K-, and 64K-byte PROMs supported by STD-DOS. The RC708 Interface Cable connects the PROM programmer to the PC.

SYSTEM COMPONENTS

The STD-LIB package consists of three components that comprise the documentation, programs, and files. Use the following list to ensure your package is complete.

- STD-LIB Library Reference manual in a loose-leaf binder. Check the manual against the table of contents to be sure you have all the pages.
- STD-LIB source disks. These disks contain the source code for all the library modules. These are files ending in .ASM with names similar to the library commands. For example, the source code for initializing the 7508 card is contained in a file called 7508INIT.ASM. The disk also contains STD-LIB.INC, the Assembly language source code include file. The directive STD-LIB.INC is placed in the code that generates new library modules. This file contains software required by library modules, including C and BASIC linkages and library macros such as MODREG. The last file on the source disk is ASMLIB.BAT. This is a batch file that is used to assemble an object module for each of the different languages supported by the library and then add the object code to the different library versions.
- STD-LIB object disk. This disk contains the library files that are ready to link to your application program:

STDLIBCS.LIB (small model C)
STDLIBCM.LIB (medium model C)
STDLIBCL.LIB (large model C)
STDLIBB.LIB (BASIC)
STDLIBAL.LIB (Assembly)

WARRANTY REGISTRATION CARD

A postage paid warranty registration card is included as part of this package. This card must be filled out to receive information on updates and enhancements. This card should be filled out and mailed immediately after you read the software end-user agreement (located at the front of this manual).

HOW TO USE THIS MANUAL

Section 1, Introduction, is intended for the user who is not familiar with one or more of the aspects of the library. This includes those unfamiliar with the PC, STD-DOS, and programming languages. This section leads you through an example session. Section 1 can be skipped by users who are familiar with all aspects of the library.

Section 2, Operation and Programming, describes how to write application programs for the library. This section assumes you are familiar with the PC, STD-DOS, and programming in your application language.

Section 3, Creating New Library Routines, describes how to add new functions to the library. The specification for library modules in this section was used to create the modules already in the library, so this section also provides insight into the details of how the library works.

Section 4, C Language Library, describes the library modules that can be called by application programs written in small, medium, and large model C.

Section 5, BASIC Language Library, describes the library modules that can be called by application programs written in Compiled BASIC.

Section 6, Assembly Language Library, describes the library modules that can be called by application programs written in Assembly language.

GETTING STARTED

To make full use of the library, you should be familiar with the operation of the PC, STD-DOS, and your assembler or compiler. You should know how to program in the language you have chosen to use. If you are not familiar with all these things, they can be learned from the user's manuals of the products. Further sources of information are also listed in the References section of this manual. Perhaps the easiest way to learn if you are not familiar with these things is to try the tutorial that follows. The tutorial is structured to lead you through all steps involved in using the library, even if you don't understand everything. The tutorial gives you an idea of what you need to learn about components used by the library.

Backing up the disks

Before doing anything, you should make backup copies of your two STD-LIB disks. You should also make frequent backups of your application program disk. To make a copy of a disk in a 2-drive system, do the following:

Put the PC-DOS disk in drive A. Boot the PC if it is not already at the prompt ("A>"). To return to DOS if your computer is on, simultaneously press and hold the Ctrl key, the Alt key, and the Del key. To get to DOS if your PC is turned off, turn the PC on with the PC-DOS disk in drive A.

Type DISKCOPY A: B: followed by the enter key. The screen should respond with something similar to:

A> DISKCOPY A: B:

Insert source diskette in drive A:

Insert target diskette in drive B:

Press any key when ready . . .

Remove the DOS disk from drive A. Put the disk you want copied (the "source" disk) in drive A. Put the disk you want to copy the information to (the "target" disk) in drive B (if this disk contains information, it will be erased and replaced by this backup).

Press any key. When the copying is finished, the computer will ask whether you want to copy another. Type y the first time so that you can copy both STD-LIB disks. Use a second target disk for the backup of the second library disk. Type n the second time the PC asks if you want to copy another.

It is good practice to place write protect tabs over the notches in your backup copies to prevent them from accidentally being written to. With a felt-tip pen, gently label the disks and put them away in a safe place. These backup copies should be kept in a different location from the main copies.

The procedure for disk backup in a 1-drive system is similar to that just explained, refer to your PC-DOS manual for the details.

TUTORIAL

The easiest way to understand how the library works is to walk through the steps of creating and executing an application program that uses library routines. This tutorial assumes that you have a PC with two disk drives called A and B. If your PC doesn't have these same drives, use the PC-DOS manual to modify the tutorial to match the drives in your system.

STEP 1: Define the application program and write a specification. As an example, let us assume that our demonstration program initializes an STD I/O card (mapped in its as-shipped condition), and causes something visual to occur at the card. This program assumes that the PC is still attached to the STD-DOS system when the program is executed. This allows the results of the initialize to be printed on the screen of the PC.

STEP 2: Select the STD BUS I/O cards to be used in the end system. We will initialize a Pro-Log 7303 Keyboard Display Card and write a message to its alphanumeric displays. If you have this card, plug it into your STD-DOS system with the ejector tab at the top. You can still do the tutorial if you don't have the card, your demonstration program will write a message to the PC screen that it couldn't find the 7303 card. The 7303 card is mapped at I/O ports D0 and D1 as shipped, remove any card in your system (other than the 7303) that is mapped at these addresses.

STEP 3: Write any new library routines you need for cards not currently supported by the library. Let's assume that we don't need any new library routines.

STEP 4: Select the language in which the application program is to be written. For demonstration purposes, we will show three languages; small model C, BASIC, and Assembly. You should only do the demo in the language that you intend to use for your application programs. If the demo operation to be performed is language dependent, the operation(s) for each language is provided. The language-dependent operations are preceded by the name of the language, in parentheses, followed by the steps to be performed by you for that language. When the operation involves typing commands, do **not** type the language name in parentheses that identified the language to which the operations applied.

(C) The C Compiler comes on three separate disks. The tutorial assumes that you have transferred your small model C compiler files onto two new disks: a compile disk and a link disk. This can be done by copying files to two new blank disks as outlined in the following. Refer to your PC manual if you need information on the COPY command.

Create a CONFIG.SYS file. This can be done by using EDLIN as described in STEP 5. The CONFIG.SYS file should contain the following statements:

```
FILES = 10  
BUFFERS = 10
```

To read the files from the C Compiler disks, the path must be defined. This can be done by typing:

```
PATH A:\BIN  
SET INCLUDE=A:\INCLUDE  
SET LIB=A:\LIB  
SET TMP=B:\
```

C Compiling disk:

Copy from disk 1 of 3:

BIN\MSC.EXE
BIN\P0.EXE
BIN\P1.EXE
BIN\P2.EXE
BIN\P3.EXE

Copy from disk 2 of 3:

INCLUDE\ASSERT.H
INCLUDE\CONIO.H
INCLUDE\CTYPE.H
INCLUDE\DIRECT.H
INCLUDE\DOS.H
INCLUDE\ERRNO.H
INCLUDE\FCNTL.H
INCLUDE\IO.H
INCLUDE\MALLOC.H
INCLUDE\MATH.H
INCLUDE\MEMORY.H
INCLUDE\PROCESS.H
INCLUDE\SEARCH.H
INCLUDE\SETJMP.H
INCLUDE\SHARE.H
INCLUDE\SIGNAL.H
INCLUDE\STDIO.H
INCLUDE\STDLIB.H
INCLUDE\STRING.H
INCLUDE\TIME.H
INCLUDE\V2TOV3.H
INCLUDE\SYS\LOCKING.H
INCLUDE\SYS\STAT.H
INCLUDE\SYS\TIMEB.H
INCLUDE\SYS\TYPES.H
INCLUDE\SYS\UTIME.H

C Linking disk:

Copy from disk 3 of 3:

BIN\LINK.EXE
BIN\LIB.EXE

Copy from disk 2 of 3:

LIB\SLIBC.LIB
LIB\SLIBFP.LIB
LIB\EM.LIB

STEP 5: Write the application program. Let's use EDLIN to write this program.

With your PC-DOS disk in drive A, return to the DOS prompt on your PC ("A>") the same as you did when you made the backup copies of the library disks. Place a blank disk that will contain your demo program in drive B. Format the disk in drive B by typing **FORMAT B:**. To start EDLIN, type:

```
(C)                EDLIN B:DEMO.C
(BASIC)            EDLIN B:DEMO.BAS
(ASSEMBLY)         EDLIN B:DEMO.ASM
```

The following message and prompt are displayed:

```
New file
*
_
```

Let's begin entering text by entering an **I** command to insert lines.

```
(C)
extern pl7303_init();
extern pl7303_word();

main()
{
    int baseadr;
    int error;
    baseadr = 0x1D0;
    if ((error = pl7303_int(baseadr))==-1)
        printf("no 7303 installed at address D0");
    if (error==0)
        printf("7303 installed \n");
    pl7303_word("hello")
}
```

(BASIC)

```
10 BASEADR%=&H1D0
20 CALL PL7303INIT(ERROR%,BASEADR%)
30 IF ERROR%=0 THEN PRINT "7303 INSTALLED"
40 IF ERROR%=-1 THEN PRINT "NO 7303 INSTALLED AT ADDRESS D0"
50 MSG$="HELLO"
60 CALL PL7303WORD(ERROR%,MSG$)
70 END
```

(ASSEMBLY)

```

EXTRN      PL7303_INIT:FAR          ;define external routines
EXTRN      PL7303_WORD:FAR          ;

CODE       SEGMENT                  ;start of segment
ASSUME     CS:CODE,DS:CODE,ES:CODE,SS:STACK
MOV        AX,CS                    ;set ES & DS = DS (pl7303_word gets ascii
MOV        DS,AX                    ;
MOV        ES,AX                    ;from ES, DOS from DS)
MOV        BX,[BASEADR]             ;set BX = 7303 base port
CALL       PL7303_INIT              ;call 7303 initialize
CMP        AX,-1                    ;see if err code returned
JE         DEMOERR                  ;if error found, go
LEA        DX,HELLOMSG              ;set DS:DX = "HELLO" msg
CALL       PL7303_WORD              ;call 7303 word
LEA        DX,NOERRMSG              ;set DS:DX = no msg adr
EXITDEMO:  MOV        AH,09H         ;09 = DOS disp string func
INT        21H                     ;execute interrupt 33
MOV        AH,4CH                   ;4C = terminate func
INT        21H                     ;execute interrupt 33
DEMOERR:   LEA        DX,ERRMSG      ;set DS:DX = error msg adr
JMP        EXITDEMO                ;go show msg & exit
HELLOMSG   DB         'HELLO'       ;PL7303_WORD requires
DB         00H                     ;string null terminator
NOERRMSG   DB         '7303 INSTALLED $' ;DOS string requires "$" terminator
ERRMSG     DB         'NO 7303 INSTALLED AT ADDRESS D0 $' ;
BASEADR    DW         0100H         ;base port adr for 7303

CODE       ENDS                    ;end of code segment
STACK     SEGMENT      STACK        ;start of stack segment
DB        80H DUP (?)              ;reserve 80 bytes for stack
STACK     ENDS                    ;end of stack segment

END                                           ;end of program directive

```

(ALL LANGUAGES)

When you finish typing in the program, press the Ctrl key simultaneously with the Break key. This discontinues the insert mode of operation.

If you make a mistake in entering the demo program, type in the line number of the line with a mistake. The current line is then displayed followed by a blank line to enter what you wanted. There is a complete description on the operation of EDLIN in your PC-DOS manual if you need more information.

Save the demo program you have entered on the disk and leave EDLIN by typing E to end edit. Your demo will be saved on drive B with the name DEMO.C, DEMO.BAS, or DEMO.ASM.

Remove the disk from drive A and replace it with the library object file disk. Type in the following to put the library routines on the same disk as the demo program:

```
COPY STDLIBCS.LIB B:  
COPY STDLIBB.LIB B:  
COPY STDLIBAL.LIB B:
```

Remove the library disk from drive A and replace it with the PC-DOS disk.

STEP 6: Compile or assemble the application program.

(C) To compile the demo program that has been written in C, place the C compiling disk that you created in drive A.

Change over to drive B as the default drive by entering:

```
B:
```

Invoke the compiler by typing:

```
A:MSC DEMO;
```

Type A: followed by a carriage return to change back to drive A as the default.

(BASIC) To compile the demo program that has been written in BASIC, place the Microsoft Compiled BASIC disk containing BASCOM.EXE in drive A. Change over to drive B as the default drive by entering:

```
B:
```

Invoke the compiler by typing:

```
A:BASCOM DEMO;
```

Type A: followed by a carriage return to change back to drive A as the default.

(ASSEMBLY) To assemble the demo program that has been written in assembly language, place the Microsoft Macro Assembler disk containing MSC.EXE in drive A. Invoke the assembler by typing:

MASM B:DEMO;

STEP 7: Link the compiled or assembled application program with the library version in the language that matches the application program.

For the demo, we need to link the compiled application program, DEMO.OBJ, with the appropriate language version of the library. This is done by typing:

(C) Remove the C compiling disk from drive A and replace it with the C linking disk you created.

(C)	LINK B:DEMO.OBJ,,,B:STDLIBCS.LIB
(BASIC)	LINK B:DEMO.OBJ,,,B:STDLIBB.LIB
(ASSEMBLY)	LINK B:DEMO.OBJ,,,B:STDLIBAL.LIB

STEP 8: Connect the STD-DOS system to the PC.

Connect the power supply to the STD BUS card rack. The connectors on the power supply and card rack are polarized and locking.

Connect the end of the interconnect cable with the D-connector to the IBM PC COM1 serial port at the back of the PC. (If COM2 must be used, refer to the STD-DOS User's Guide.)

Connect the other end of the interface cable to Connector J1 on the 7863 card (the 26-pin transition connector on the front of the card).

The power supply has a switch for use with different line voltages. Be sure the switch is set to the appropriate line voltage. Be sure the power supply is turned off and plug it in. Power on the STD BUS.

STEP 9: Use the STD utility on the STD-DOS UTILITIES disk to download the .EXE file created by the link to the STD BUS drive B.

With the PC-DOS prompt (A>) displayed, remove the disk from drive A and replace it with the STD-DOS Utilities disk. Start the STD-DOS utility program "STD" by typing STD followed by a carriage return (enter). This should result in the following display on the PC screen:

STD version x.xx
(c)Copyright 1985 Pro-Log Corp.

STD A>

The STD-DOS system is functioning correctly if the prompt "STD A>" is displayed on the PC.

To download the DEMO.EXE file to the STD BUS, Press the Alt key simultaneously with the F3 key. This begins the download utility. You should see the following display on the screen:

STD A> Copy from PC to STD BUS --
PC filename:

Type in **B:DEMO.EXE** followed by a carriage return. The screen should respond with:

STD directory:

Type in **B:**. The screen should respond with:

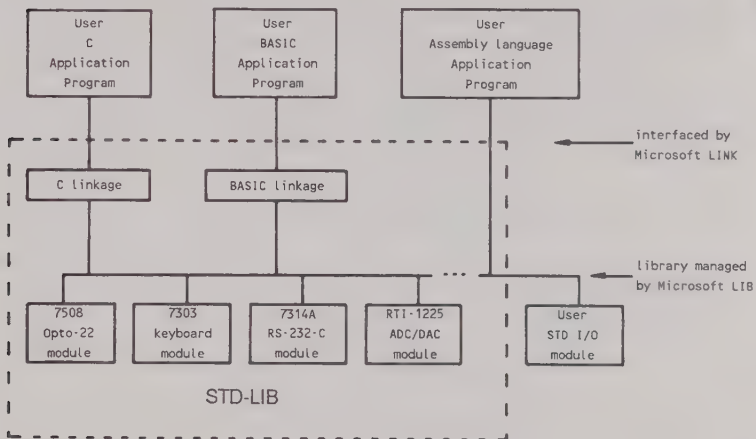
Copying file(s) ...
1 file(s) copied

(BASIC) The file BASRUN.EXE must also be downloaded from the Compiled BASIC disk. This file is required to execute a BASIC program. Remove the DEMO disk from drive B, and replace it with the BASIC disk. Repeat the download process to STD drive B with the file B:BASRUN.EXE.

STEP 10: Execute the application program. Cause the program to execute in the STD BUS system by typing in its name, **DEMO**, while the STD prompt is displayed on the PC screen. This should result in a message on the PC screen saying either "7303 INSTALLED" or "NO 7303 INSTALLED AT ADDRESS D0". If the 7303 INSTALLED message was displayed, the message HELLO should be written to the first five alphanumeric displays of the 7303 STD card followed by three blank displays.

FUNCTIONAL DESCRIPTION

The following figure shows the library operation as a block diagram. This provides an overview of the relationship between the library and user application programs. The paragraphs following describe each of the blocks of the figure in more detail.



USER APPLICATION PROGRAM. The user application program can be written in Microsoft C, Compiled BASIC, or assembly language. The application program is linked by LINK with the version of STD-LIB that is in the same language. LINK searches the library to resolve the application programs' external references to routines and variables. When LINK finds such a routine, it extracts a copy of the library module and links it to the application program.

The application program must first call the library initialize program for each user card supported by the library, to tell the library routines how the card has been configured. The application program can then call the library routines an unlimited number of times. A single copy of a library routine for a card supports more than one card of that type in most cases.

C LINKAGE. The user is provided with separate libraries for the three compilation models (small, medium, and large) available for Microsoft C.

BASIC LINKAGE. The STD-LIB BASIC library supports Microsoft Compiled BASIC.

MICROSOFT LINK INTERFACE. The Microsoft linker (LINK), is used to combine the user application programs with the STD-LIB library routines accessed by the application programs. The library routines are linked in by including the name of the version of the library that matches the application program language in the LINK command line. This can be done during the standard link operation used to create an executable program. LINK is included as part of STD-DOS.

MICROSOFT LIB LIBRARY MANAGER. The Microsoft library manager, LIB, is used to organize and maintain STD-LIB. The library consists of object modules that are assembled or compiled instructions or data ready for linking. The user does not require LIB to use STD-LIB or to upgrade with future versions of STD-LIB. Microsoft LIB is required only if users wish to add their own library routines to STD-LIB or to modify existing routines. Microsoft LIB comes as a part of the Microsoft Macro Assembler.

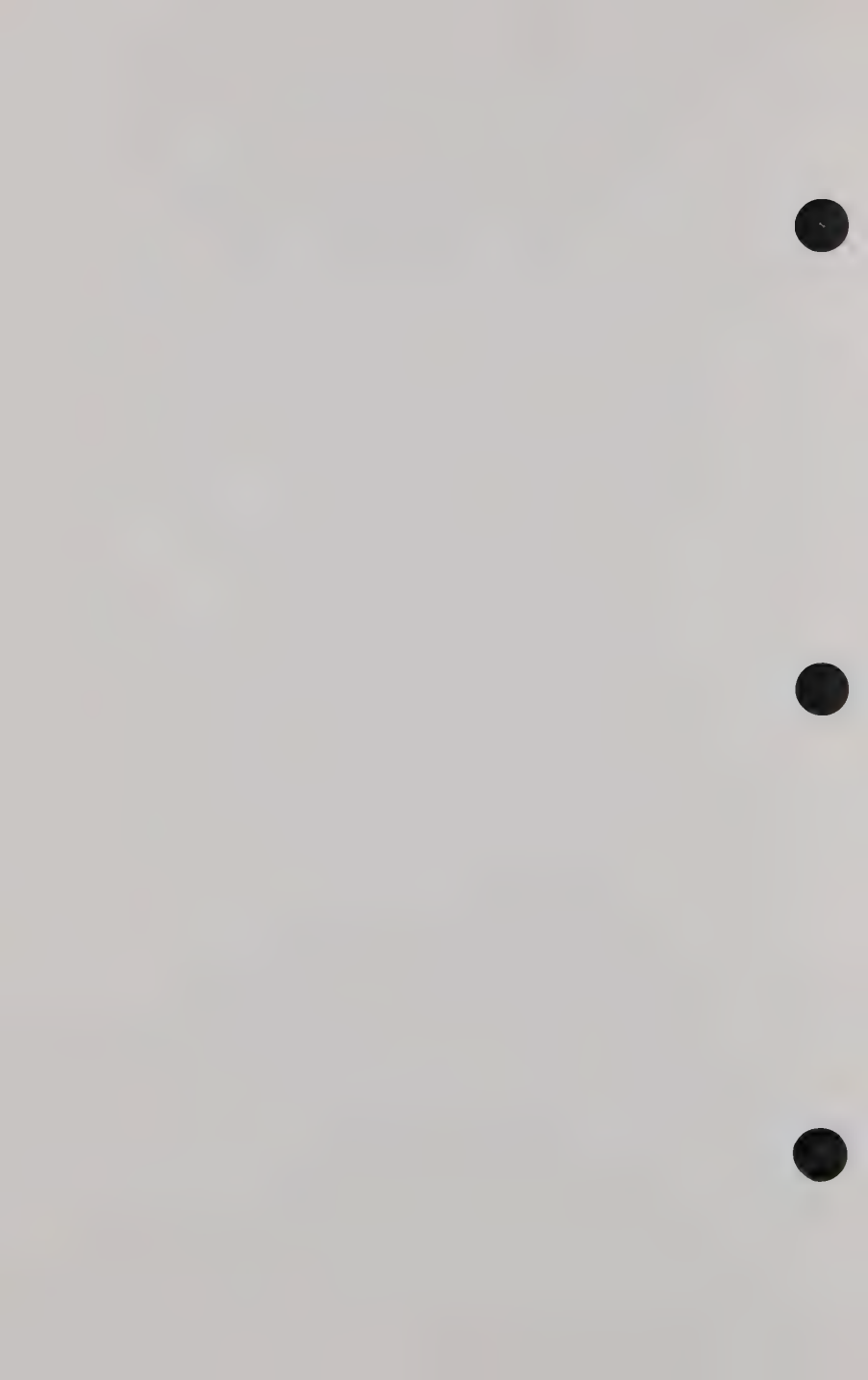
I/O CARD MODULES. The core of the library is the I/O card modules. Each module consists of a number of assembly language routines that are all related to a single STD I/O card or family of cards. An assembly language application program calls these routines directly. The BASIC and C language versions of STD-LIB were built by using the same assembly language routines combined with linkage routines that interface with BASIC or C. The fact that the core of each language version of the library is the same lets the user change languages with complete confidence should the need arise.



Section 2

OPERATION AND PROGRAMMING

CONTENTS	PAGE
Introduction.....	2-1
The Procedure	2-1
Writing the Application Program.....	2-2
Linking an Application Program with STD-LIB.....	2-3
I/O Card Mapping	2-5
I/O Card Initialization.....	2-7
Parameter Names.....	2-7
Numeric Representation	2-7



OPERATION AND PROGRAMMING**INTRODUCTION**

This section describes how to use STD-LIB modules with your application programs. The application program is linked with the version of STD-LIB that matches the language of the application program.

To use a module within an application program, set up the parameters that are required by the module and then call the module. Each module is called in a single program statement.

To understand the operation and structure of the library modules, refer to Section 3, Creating New Library Routines. You need not understand how the library was generated to make use of it.

THE PROCEDURE

In making use of STD-LIB, you will find yourself following the same basic steps each time you write an application program. This procedure can be summarized as follows:

1. Define the application problem and write a specification.
2. Select the STD BUS I/O cards to be used in the end system.
3. Write and debug any new library routines you need for cards not currently supported by the library, following the specifications of Section 3, Creating New Library Routines.
4. Select the language in which the application program is to be written. Take into consideration the training of the programmers, the amount of system memory, the speed of execution, time allowed for program development, and complexity of the application.
5. Write the application program in C, BASIC, or Assembly language. Write the program on the PC using a text editor such as BRIEF (or a line editor such as EDLIN). Save the application program on disk under a file name ending in .ASM.

6. Compile or assemble the application program.
7. Link the compiled or assembled application program with the library version in the language that matches the application program. If you are writing in assembly language, you may wish to create a link map for use during debug.
8. Connect the STD-DOS system to the PC. Power on the STD BUS.
9. Use the STD utility on the STD-DOS UTILITIES disk to download the .EXE file created by the link to the STD BUS drive B.
10. Execute the application program. Debug the program against the specification generated in Step 1 using DEBUG, and the map file generated by the link in Step 7. Go back to Step 5 until the application program has no bugs.
11. Use the STD-DOS ROMDISK utility to place the application program .EXE file into ROM in the target STD-DOS system.

WRITING THE APPLICATION PROGRAM

Write your application program just as you would write any normal program, in either Microsoft C, BASIC, or assembly language.

Refer to the module descriptions in the section of this manual that corresponds to the language you are using. These descriptions completely define the available commands and give examples of how to use them. Call the module routines as often as you like following the structure given in module definitions.

The module descriptions all have a section called "structure." Structure indicates the types of instructions that must be included somewhere in the application program that calls the module. The instructions don't have to be identical to those shown defined by structure, just of that type. Structure also gives the format of the command itself in bold type face.

The "structure" (format) defined for C modules include instructions that declare references that are external to the application program (extern) and declarations of variable type (int and char). The structure for Assembly programs include instructions that declare references that are external to the application program (EXTRN), and FAR declarations.

The BASIC programs use "%" to designate integer variables, these can be omitted with the proper use of the DEFINT statement.

All strings passed to assembly language library modules must be terminated with a null character (00H).

LINKING AN APPLICATION PROGRAM WITH STD-LIB

After your application program has been compiled or assembled, it is combined with the STD-LIB files it accessed and converted to an executable (.EXE) output file by Microsoft LINK. The linker is shipped as part of Pro-Log's STD-DOS and can also be found in the Microsoft Macro Assembler. The following description provides an overview of how to use LINK. For more detailed information, use the manual shipped with the LINK program.

LINK can be used in three different ways: through a DOS command line, in response to prompts, or with a response file.

Creating an Executable Program

The DOS command line has the form:

```
LINK[option] object-file,[exe-file],[map-file],[lib-file]
```

The **options** are execution options that control the operation of link.

The **object-file** is the name or names of the user application object files you want to link together. The files must have been created using MASM, the Microsoft C Compiler, or the Microsoft BASIC Compiler.

The optional **exe-file** is the name of the executable file you wish to create. LINK uses a default name if you do not supply one (the name will be the same as the object file with the extension .EXE).

The optional **map-file** is the name of the file to receive the map listing.

The **library-file** is the name or names of the libraries containing routines you wish to link to your program. The STD-LIB name should be selected from the following table to match the language of your application program:

STD-LIB VERSION	APPLICATION PROG. LANGUAGE
STDLIBCS.LIB	C (small model)
STDLIBCM.LIB	C (medium model)
STDLIBCL.LIB	C (large model)
STDLIBB.LIB	BASIC
STDLIBAL.LIB	ASSEMBLY

More than one library can be specified in this command line to allow you to use other libraries in addition to STD-LIB. If you are specifying more than one library, separate the names by spaces or plus signs (+).

An example of a typical command line is:

```
LINK USERPROG.OBJ,,,STDLIBAL.LIB
```

Refer to the LINK documentation for information on using LINK in response to prompts or with a response file.

I/O CARD MAPPING

STD-DOS systems have reserved locations 100-1EF in the I/O map for customer use. To prevent future compatibility problems, all customer-installed I/O cards should be mapped in this area.

All cards that support only 8-bit I/O addressing should be treated in software as if they decoded 10-bits of address and were mapped in the range 100-1EF. Future Pro-Log products will use the IOEXP line to indicate accesses to this area of the I/O map to prevent 8-bit cards from responding to multiple locations in the I/O map.

STD-LIB DEFAULT I/O MAP

HEX RANGE	USAGE
100-103	Only 10-bit address I/O cards
104-138	For customer use
139	Only 10-bit address I/O cards
13A	For customer use
13B-13D	Analog Devices RTI-1260 default
13E-13F	For customer use
140-141	Pro-Log 7000 software default
142-14F	For customer use
150-153	Pro-Log 7507 default
150-157	Pro-Log 7508 default
158-15F	For customer use
160-167	Pro-Log 7312 and 7314A default
168-1AF	For customer use
1B0-1B7	Only 10-bit address I/O cards
1B8-1BA	For customer use
1BB-1BF	Analog Devices RTI-1225 default
1C0-1C7	Analog Devices RTI-1262 default
1C0-1CF	For customer use
1D0-1D1	Pro-Log 7303 default
1D2-1EF	For customer use

Many of the library routines allow you to affect more than one card with the same library command. This is done by "contiguous mapping." This is where cards are mapped so the port addresses of each card are followed immediately by the port addresses of the next card without any addresses being skipped.

To allow some of the library commands to affect only one of the contiguously mapped cards, the cards are identified by number. The card number of the contiguously mapped cards with lowest port address is 0, with the other cards following in ascending order. Sometimes only a part of a card is identified by a number, such as the 7312/7314A commands that affect single channels and allow up to 32 contiguously mapped channels. One card can be singled out by a library command by passing the number of the card in the parameter list.

The base address of a card is the lowest port address of the card. If the library routine knows the address of this port, it automatically knows the address of any other ports on the card. For contiguously mapped cards, the base address of the lowest port address of all the contiguously mapped cards is specified. With this address, and the

number of contiguously mapped cards, the library routines automatically know the address of all the ports for all the contiguously mapped cards.

I/O CARD INITIALIZATION

The library contains routines for initializing the I/O cards and the associated library software routines for that card. These initialization routines may perform an automatic initialization for a single card installed in the as-shipped condition. If the routine does this automatic initialization, it is not necessary to call the routine if the card meets the default conditions.

PARAMETER NAMES

Many of the parameter names used in the library are not unique. When using library routines for more than one type of card, do not define a variable name more than once.

All routines internal to the library and not intended for the user are preceded by two "at" symbols ("@@"). To prevent possible contention, do not use any names in your application program that are preceded by @@.

NUMERIC REPRESENTATION

The numeric representation used in the library definitions is that of the language for the associated library. This representation may look different from that used in the user's manual for the STD card. Hexadecimal number 100 is equivalently represented by 0x100 (C), &H100 (BASIC), and 0100H (assembly).





Section 3

CREATING NEW LIBRARY ROUTINES

CONTENTS	PAGE
Introduction.....	3-1
STD-LIB.INC Include File.....	3-2
MODREG Macro Call.....	3-2
START Macro Call.....	3-2
STRSTART Macro Call.....	3-3
FINISH Macro Call.....	3-4
DATASTART and DATAEND Macro Calls.....	3-4
DATASEG Macro Call.....	3-4
Module Source Code.....	3-5
Example Module Structure.....	3-6
Application Program Calls.....	3-6
Parameter Passing.....	3-7
Generating STD-LIB Language Versions.....	3-10
Example Source Code Listing.....	3-12
Example Small Model C Language Object Code Listing.....	3-13
Example BASIC Language Object Code Listing.....	3-15
Example Assembly Language Object Code Listing.....	3-17
Returned Value.....	3-19
General Practices.....	3-19
Documenting New Modules.....	3-20

CREATING NEW LIBRARY ROUTINES

INTRODUCTION

This section describes how to add new modules to STD-LIB and contains the library specification. Since the library was generated by following this procedure, this section also describes the structure of STD-LIB. This section may be skipped by users who do not require such information.

New modules can be added to the library by following the STD-LIB specification contained in this section. The Microsoft Macro Assembler version 4.0 (which contains LINK and LIB) is required to add new routines to the library.

New routines are written in 8088 assembly language. The complex STD-LIB requirements of interfacing with several programming languages is handled by STD-LIB macros.

The assembly language program for generating a new module must include the following structure:

```
INCLUDE STD-LIB.INC
DATASTART
:
:
DATAEND

MODREG <register list>
START cardname,funcname,<registerlist>,
number
DATASEG register
:
:
FINISH cardname,funcname
```

STD-LIB.INC INCLUDE FILE

The **INCLUDE STD-LIB.INC** directive added to a library module routine inserts source code from the **STD-LIB.INC** source file into the library module routine during assembly.

STD-LIB.INC contains the software for the C and BASIC linkages and various macros for **STD-LIB**. Much of this software contains conditional statements that become a part of the assembled module for only one of the supported languages. The functions of these macros include adding the underscore between the card name and function name for the C and assembly library calls.

MODREG MACRO CALL

The **MODREG** macro flags the registers to be saved by the **START** macro. The register list defines the registers used by the module routine. The **START** macro causes these registers to be saved on the stack in the assembly language routine, but not saved for any of the other languages.

START MACRO CALL

The **START** macro defines the module. The routine name is defined in two parts: the card name (cardname) and the function name (funcname). These combined names cannot exceed 29 characters in length. Both cardname and funcname must be lower case and consist only of numbers and letters.

Cardname is always 6 characters in length. The first 2 characters identify the manufacturer of the I/O card associated with the routine (e.g., "pl" for Pro-Log). The next four digits are the manufacturer's model number (e.g., "7314" for the Pro-Log serial communication card).

Funcname indicates the type of operation performed by the routine (e.g., "init" for an initialization routine).

The register list defines the registers used by the module routine for parameters that are passed to it. The return parameter is always register AX and is not defined in this list. The allowed registers for general parameter passing are AX, BX, CX, DX, SI, and DI. Any of the 8-bit registers that comprise the mentioned registers are also allowed. The register list must list registers in uppercase. Refer to the subsection on parameter passing in this section for details on how to select the registers used for parameter passing.

Number defines the scratch pad area for the module routine. This is the number of bytes reserved on the stack every time the program is called. Within the module routine, the first scratch byte is accessed as [BP-1] and the second byte or the first word of scratch are accessed as [BP-2]. This scratch area is removed from the stack at the end of the module routine call. Allocating this storage area on the stack allows this routine to be interrupted.

The START macro can't be used for passing character strings. There is a special macro that allows a string to be passed, the STRSTART macro. This macro is very similar to the START macro.

STRSTART MACRO CALL

The STRSTART macro is placed in the module source code where START would be placed, following the same format:

STRSTART cardname,funcname,<registerlist>,number

STRSTART is defined exactly as START is, except for the details on parameter passing. Refer to Parameter Passing in this section for details on how to select the registers used for parameter passing when a string is involved.

FINISH MACRO CALL

The finish macro defines the end of the module routine. When this macro is encountered, all register contents saved by the START macro are restored. At this location, the STD-LIB.INC file inserts more source code to allow a proper return to the calling program.

DATASTART AND DATAEND MACROS CALLS

Global or local storage can be defined within a library routine with a data declaration (DB, DW, DD, DQ, DT). The data declaration must be placed between the DATASTART and DATAEND macro calls. This storage may be declared PUBLIC in the library routine. The storage can be accessed from an assembly language calling program by referring to the storage in the calling program with an EXTRN statement.

The DATASTART and DATAEND macros are required to access global variables defined in other modules, even if this module is not declaring storage. This is a requirement of the DATASEG macro that sets the segment register for the variables. All external variables for a library routine must be declared within the DATASTART/DATAEND macro pair.

DATASEG MACRO CALL

The DATASEG macro sets the segment register that accesses the data area defined within the DATASTART and DATAEND macros. Register is either the ES or DS register, where the register is defined in uppercase. The DATASTART and DATAEND macros must be invoked prior to calling DATASEG.

The DATASEG macro changes the contents of the DS register. If the DS register is used to pass a parameter, the contents of the DS register should be saved before the DATASEG macro is invoked.

MODULE SOURCE CODE

After filling in the module structure, a general assembly language routine is written for the STD card. This source code is turned into object modules for each of the different languages supported by the library. This is accomplished through the conditional assembly

statements of the macros you include in your source code. The ASMLIB.BAT file automatically generates the different object modules and adds them to the different library versions. If the description in this section still leaves you uncertain about how to write source code for a library module:

1. Find the module in the library that most closely resembles the module you wish to write.
2. Print out the source code from the .ASM file.
3. Use the printout as an example.

EXAMPLE MODULE STRUCTURE

The following is an example of a typical structure that could be used for the Pro-Log 7508 card initialization routine (like the one actually included in the library modules).

```

INCLUDE STD-LIB.INC

PUBLIC  @@NUMBER_OF_RACKS      ;public
PUBLIC  @@7508_BASE_PORT      ; parameters
PUBLIC  @@CUR_7508_DATA        ;

DATASTART                      ;
@@NUMBER_OF_RACKS DW 2         ;default=2
@@7508_BASE_PORT DW 0150H      ;default=150
@@CUR_7508_DATA DB 88 DUP (00H);port data
DATAEND                        ;

MODREG  <DX,BX,ES>             ;
START   pl7508,init,<DX,BH>,0;

        DATASEG ES             ;
        .
        .
        .
FINISH   pl7508,init           ;
        END                    ;

```

APPLICATION PROGRAM CALLS

When the routine is called from an application program, both the card name and function name defined in the START macro must be used. The STD-LIB interface program was written so that C and assembly language routines must separate the two names with an underscore to improve readability. Microsoft BASIC does not allow an underscore, so the two names are joined. For example, the assembly language library routine with the card name "pl7508" and the function name "rackpwr" is a routine for the Pro-Log 7508 I/O card that turns on or off rack power (rackpwr). This routine is called from C as pl7508_rackpwr, from BASIC as PL7508RACKPWR, and from assembly language as PL7508_RACKPWR.

PARAMETER PASSING

The actual registers used by a library routine are defined by the register list in the START or STRSTART statement. The allowed registers for general parameter passing are AX, BX, CX, DX, SI, and DI. Any of the 8-bit registers that comprise the mentioned registers are also allowed. The contents of these registers are returned intact for assembly language programs. However, the AX register is overwritten by the returned parameter.

When passing a memory address as a parameter (for parameters other than strings), the address must be passed as an offset address and a segment address. When a memory address is passed, one of the allowed general registers contains the memory offset address. This register must be followed immediately by either DS or ES in the register list to contain the segment address. Thus, the maximum number of addresses that may be passed is two.

One value is always returned from a routine in the AX register. In BASIC, this parameter is always first in the argument list. In C, the function itself takes on the returned value (e.g., result = function(arguments);).

You may return additional values from a routine by passing one or two addresses to the routine. The address is passed in the argument list (in segment/offset form) and is the location where the returned value(s) is to be placed. The address may be a single memory location or the first

location of an array of returned values. The following example shows fragments of a library routine returning five parameters:

```

START      ex0000,ample,<BX,ES,SI,DS>
PARAM1     DW 1234
           .
           MOV    DS:[SI], PARAM1
           .
           .
DATASEG     DS
           .
           .
           MOV    ES:[BX], PARAM2
           MOV    ES:[BX+2], PARAM3
           MOV    ES:[BX+4], PARAM4
           MOV    AX, PARAM5
FINISH      ex0000,ample

```

Note that some care must be used if you are passing a pointer using a segment register if you define a local data segment, since the DATASEG macro changes the register value, thus losing one of your original parameters.

The following example application programs show how the preceding example library module could be called.

From C:

```

int ex0000__ample();
int array[3];
int value1;
value5=ex0000__ample(&array[0],&value1);

```

From BASIC:

```

DIM    VALUE1%(1)
DIM    ARRAY%(3)
CALL   EX0000AMPLE(VALUE5%,ARRAY%(0),VALUE1
%(0))

```

From Assembly:

ARRAY	DW	3 DUP (?)
VALUE1	DW	?
	.	
	.	
	MOV	BX,SEG ARRAY
	MOV	ES,BX
	LEA	BX,ARRAY
	.	
	.	
	MOV	SI,SEG VALUE1
	MOV	DS,SI
	LEA	SI,VALUE1
	CALL	EX0000_AMPLE

If a string is to be passed to a library routine, that routine must contain the STRSTART macro. The address of the memory location pointing to the string is passed in the same way addresses are passed to the START macro, except that only the ES register can be used. Only one string may be passed as a parameter. A memory address may also be passed to STRSTART, and is distinguished from the string address because it must use the DS register.

The STRSTART macro scans the string passed to it by C and Assembly routines for a null (00) terminator and generates a count of string length. The null terminator is an automatic function of C, but must be placed at the end of an Assembly string by the application program. The string length is placed in register AX, so the AX register can't be used to pass parameters to the library routine. The STRSTART macro gets the string length for strings passed by BASIC from the string description that is part of the string passed in BASIC language calls.

GENERATING STD-LIB LANGUAGE VERSIONS

The STD-LIB that is linked in with an application program comes in different versions that match the language of your application program. The different STD-LIB versions are shown in the following table:

STD-LIB VERSION	APPLICATION PROG. LANGUAGE
STDLIBCS.LIB	C (small model)
STDLIBCM.LIB	C (medium model)
STDLIBCL.LIB	C (large model)
STDLIBB.LIB	BASIC
STDLIBAL.LIB	ASSEMBLY

When new modules are added to STD-LIB, these different language versions of STD-LIB must be regenerated. These different language versions can be generated automatically by the ASMLIB.BAT batch file. This command line has the form:

ASMLIB asm-file STDLIB

The asm-file is the name of the file containing the user module source code. This command line generates .OBJ files for the five language versions and then adds these files to the five library versions. This command line can also be used to change (update) a file that is already in the library. If the five .OBJ files generated by ASMLIB.BAT have the same name as files already in the library, the old .OBJ files are replaced by the new ones.

The batch file uses the Macro Assembler /Dsymbol option. This option directs MASM to define a symbol that can be used during the assembly as if it were defined in the source file. These symbols are evaluated during assembly by IFDEF and IFNDEF statements contained in the macros of the STD-LIB.INC include file. The batch file generates statements similar to the following to create the different language versions of the library:

MASM 7508INIT /DASM

MASM 7508INIT /DBASIC
MASM 7508INIT /DCSMALL
MASM 7508INIT /DCMEDIUM
MASM 7508INIT /DCLARGE

The following object file listings were generated by the first three of the preceding Macro Assembler commands. These listings provide an example of the language-dependent code generated by the library macros. A "C" character in the listing indicates a line from an include file. A single-digit character indicates the macro expansion nesting level (a "+" indicates levels greater than 9). These numbers mark the conditional code that was automatically generated by the library macros for the language. The source file that the Macro Assembler used to generate these object file listings also follows.

EXAMPLE SOURCE CODE LISTING

page 80,132

```

;*****
;*
;* pl7508_init
;* This module initializes the 7508 or 7507 configuration data
;* There are two parameters to configure, CARD_BASE_PORT and
;* NUMBER_OF_RACKS. All 7507 and 7508 cards must be mapped contiguously
;* beginning at the CARD_BASE_PORT, all 7507 cards occupy 4 ports
;* and 7508 cards occupy 8 port addresses.
;* The NUMBER_OF_RACKS parameter may be determined by the
;* number of 7507 and 7508 cards. The 7507 drives 1 rack and the
;* 7508 drives either 1 or 2 racks.
;* This routine resets all racks to a known state, power off
;* and all rack outputs set low.
;*
;* inputs: NUMBER_OF_RACKS = BX
;*         7508_BASE_PORT = DX
;*
;* output: AX = 0 ;base port address and number of racks are initialized*
;*
;*****

INCLUDE STD-LIB.INC

public @NUMBER_OF_RACKS,@7508_BASE_PORT ; public parameters
public @CUR_7508_DATA ;

DATASTART ; create static data area
@NUMBER_OF_RACKS DW 2 ; number of racks default is 2
@7508_BASE_PORT DW 0150H ; Default port address
@CUR_7508_DATA DB 88 DUP (00h) ; Current port data
DATAEND

MODREG <DX,BX,ES>
START pl7508_init,<DX,BH>,0

DATASEG DS
xor bl,bl ;
xchg bl,bh ;
mov @NUMBER_OF_RACKS,BX ; set number of racks
mov @7508_BASE_PORT,dx ; save new value for base port address
mov cx,[@NUMBER_OF_RACKS] ; number of ports to reset
; reset all 750x racks
lea bx,@CUR_7508_DATA ; get address of port data
mov_loop: mov al,0 ;
mov [bx],al ; move data into current port data
inc bx ; increment bx to next port data
out dx,al ; send data to port
inc dx ; increment port address
mov [bx],al ; move 00 to 2nd port of rack
inc bx ;
out dx,al ;
inc dx ;
mov [bx],al ; mov 00 to 3rd port of rack
inc bx ;
out dx,al ;
inc dx ;
dec al ; send ff to rack power port
mov [bx],al ;
inc bx ;
out dx,al ;
inc dx ;
loop mov_loop ;
xor ax,ax ; clear return value

exit:
FINISH pl7508_init ; restore registers and return
END

```

EXAMPLE SMALL MODEL C LANGUAGE OBJECT CODE LISTING

Microsoft (R) Macro Assembler Version 4.00

5/19/86 13:39:43

Page 1-1

page 80,132

```

;*****
;*
;* pl7508_init
;* This module initializes the 7508 or 7507 configuration data
;* There are two parameters to configure, CARD_BASE_PORT and
;* NUMBER_OF_RACKS. All 7507 and 7508 cards must be mapped contiguously
;* beginning at the CARD_BASE_PORT, all 7507 cards occupy 4 ports
;* and 7508 cards occupy 8 port addresses.
;* The NUMBER_OF_RACKS parameter may be determined by the
;* number of 7507 and 7508 cards. The 7507 drives 1 rack and the
;* 7508 drives either 1 or 2 racks.
;* This routine resets all racks to a known state, power off
;* and all rack outputs set low.
;*
;* inputs: NUMBER_OF_RACKS = BH
;*          7508_BASE_PORT = DX
;*
;* output: AX = 0 ;base port address and number of racks are initialized*
;*****

C INCLUDE      STD-LIB.INC
C              .list
C

      public    @NUMBER_OF_RACKS,@7508_BASE_PORT ; public parameters
      public    @CUR_7508_DATA                  ;

      DATASTART                                ; create static data area
1  _DATA      SEGMENT WORD PUBLIC 'DATA'
      @NUMBER_OF_RACKS DW 2                      ; number of racks default is 2
      @7508_BASE_PORT DW 0150H                  ; Default port address
      @CUR_7508_DATA  DB 88 DUP (00h)           ; Current port data

0000 0002
0002 0150
0004 0058[
00
]

005C      DATAEND
1  _DATA  ENDS

      MODREG      <DX,BX,ES>
      START       pl7508_init,<DX,BH>,0
1
1  _TEXT      SEGMENT BYTE PUBLIC 'CODE'
1  ASSUME     CS: _TEXT,DS: NOTHING,ES: NOTHING,SS: NOTHING
1  PUBLIC     _pl7508_init
1  @@DS@@      dw      ?                      ;save area for data segment
1  _pl7508_init PROC NEAR
1              PUSH BP
1              MOV  @@DS@@, DS                ;save data segment
1              MOV  BP,SP                    ;set stack frame addressability
3              MOV  DX,[BP+DISP]             ;get parameter to DX
3              MOV  BH,[BP+DISP]             ;get parameter to BH

0000 ????
0002
0002 55
0003 2E: 8C 1E 0000 R
0008 88 EC
000A 88 56 04
0000 8A 7E 06

      DATASEG      DS
1  MOV             DS,@@DS@@
1  xor bl,bl
1  xchg bl,bh
1  mov @NUMBER_OF_RACKS,BX
1  mov @7508_BASE_PORT,dx
1  mov cx,[@NUMBER_OF_RACKS]
1
0010 2E: 8E 1E 0000 R
0015 32 DB
0017 86 DF
0019 89 1E 0000 R
0010 89 16 0002 R
0021 88 0E 0000 R
1
0025 8D 1E 0004 R
0029 80 00
002B 88 07
002D 43
002E EE
1  mov_loop:      lea bx,@CUR_7508_DATA
1                  mov al,0
1                  mov [bx],al
1                  inc bx
1                  out dx,al
;
;
; set number of racks
; save new value for base port address
; number of ports to reset
; reset all 750x racks
; get address of port data
;
; move data into current port data
; increment bx to next port data
; send data to port

```

EXAMPLE SMALL MODEL C LANGUAGE OBJECT CODE LISTING (Continued)

Microsoft (R) Macro Assembler Version 4.00

5/19/86 13:39:43

Function: pl7508_init

Library: Small Model C

Page 1-2

```

002F 42                inc dx                ; increment port address
0030 88 07            mov [bx],al            ; move 00 to 2nd port of rack
0032 43                inc bx                ;
0033 EE              out dx,al              ;
0034 42                inc dx                ;
0035 88 07            mov [bx],al            ; mov 00 to 3rd port of rack
0037 43                inc bx                ;
0038 EE              out dx,al              ;
0039 42                inc dx                ;
003A FE C8            dec al                ; send ff to rack power port
003C 88 07            mov [bx],al            ;
003E 43                inc bx                ;
003F EE              out dx,al              ;
0040 42                inc dx                ;
0041 E2 E6            loop mov_loop          ;
0043 33 C0            xor ax,ax              ; clear return value

0045                exit:
                FINISH                pl7508_init        ; restore registers and return
0045 8B E5            1      MOV      SP,BP
0047 2E 8E 1E 0000 R 1      MOV      DS, @20500          ; restore data segment
004C 50              1      POP      BP
004D C3              1      RET
                1      .lall
1 _pl7508_init      ENDP
1 _TEXT            ENDS
004E                END

```

EXAMPLE BASIC LANGUAGE OBJECT CODE LISTING

Microsoft (R) Macro Assembler Version 4.00

5/19/86 13:44:05

Page 1-1

page 80,132

```

;*****
;*
;* pl7508_init
;* This module initializes the 7508 or 7507 configuration data
;* There are two parameters to configure, CARD_BASE_PORT and
;* NUMBER_OF_RACKS. All 7507 and 7508 cards must be mapped contiguously
;* beginning at the CARD_BASE_PORT, all 7507 cards occupy 4 ports
;* and 7508 cards occupy 8 port addresses.
;* The NUMBER_OF_RACKS parameter may be determined by the
;* number of 7507 and 7508 cards. The 7507 drives 1 rack and the
;* 7508 drives either 1 or 2 racks.
;* This routine resets all racks to a known state, power off
;* and all rack outputs set low.
;*
;* inputs: NUMBER_OF_RACKS = BH
;*          7508_BASE_PORT = DX
;*
;* output: AX = 0 ;base port address and number of racks are initialized*
;*
;*****

C INCLUDE STD-LIB.INC
C .list
C

public @@NUMBER_OF_RACKS,@@7508_BASE_PORT ; public parameters
public @@CUR_7508_DATA ;

DATASTART ; create static data area
1 CONST SEGMENT WORD PUBLIC 'CONST'
@@NUMBER_OF_RACKS DW 2 ; number of racks default is 2
@@7508_BASE_PORT DW 0150H ; Default port address
@@CUR_7508_DATA DB 88 DUP (00h) ; Current port data

0000
0000 0002
0002 0150
0004 0058[
00

005C
1 CONST ENDS

MODREG <DX,BX,ES>
START pl7508_init,<DX,BH>,0
1 .lall

0000
1 CODE SEGMENT BYTE PUBLIC 'CODE'
1 ASSUME CS:CODE,DS:NOTHING,ES:NOTHING,SS:NOTHING
1 PUBLIC pl7508init
1 @@@S@ dw ? ;save area for data segment
1 pl7508init PROC FAR
1 PUSH BP
4 PUSH ES
1 MOV @@@S@, DS ;save data segment
1 MOV BP, SP ;set stack frame addressability
2 MOV SI, [BP+DISP] ;get address of variable
2 MOV DX, DS:[SI] ;load DX with value
2 MOV SI, [BP+DISP] ;get address of variable
2 MOV BH, DS:[SI] ;load BH with value

0005 2E: 8E 1E 0000 R
001A 32 DB
001C 86 DF
001E 89 1E 0000 R
0022 89 16 0002 R
0026 88 0E 0000 R

002A 80 1E 0004 R
002E 80 00

DATASEG DS
1 MOV DS, @@@S@
xor bl,bl
xchg bl,bh
mov @@NUMBER_OF_RACKS, BX ; set number of racks
mov @@7508_BASE_PORT, dx ; save new value for base port address
mov cx, [@@NUMBER_OF_RACKS] ; number of ports to reset
reset all 750x racks
; get address of port data
mov_loop: mov al,0

```

EXAMPLE BASIC LANGUAGE OBJECT CODE LISTING (Continued)

Microsoft (R) Macro Assembler Version 4.00

5/19/86 13:44:05

Page 1-2

Function: pl7508&init

Library: BASIC

```

0030 88 07          mov [bx],al          ; move data into current port data
0032 43            inc bx                ; increment bx to next port data
0033 EE            out dx,al            ; send data to port
0034 42            inc dx                ; increment port address
0035 88 07          mov [bx],al          ; move 00 to 2nd port of rack
0037 43            inc bx                ;
0038 EE            out dx,al            ;
0039 42            inc dx                ;
003A 88 07          mov [bx],al          ; mov 00 to 3rd port of rack
003C 43            inc bx                ;
003D EE            out dx,al            ;
003E 42            inc dx                ;
003F FE CB          dec al              ; send ff to rack power port
0041 88 07          mov [bx],al          ;
0043 43            inc bx                ;
0044 EE            out dx,al            ;
0045 42            inc dx                ;
0046 E2 E6          loop mov_loop        ;
0048 33 C0          xor ax,ax           ; clear return value

```

```

004A                exit:
FINISH              pl7508,init          ; restore registers and return
004A 8B E5          1      MOV          SP,BP
004C 2E: 8E 1E 0000 R 1      MOV          DS, 2205222          ; restore data segment
0051 07            4      POP          ES
0052 8B 76 0C       1      MOV          SI, [BP+RETDISP]        ; get address of return variable
0055 89 04          1      MOV          DS:[SI],AX              ; store the return value
0057 5D            1      POP          BP
0058 CA 0006       1      RET          POPRET
                                .lall
                                1 pl7508init      ENDP
                                1 CODE            ENDS
END

```

EXAMPLE ASSEMBLY LANGUAGE OBJECT CODE LISTING

Microsoft (R) Macro Assembler Version 4.00

5/19/86 13:45:33

Page 1-1

page 80,132

```

;*****
;*
;* pl7508_init
;* This module initializes the 7508 or 7507 configuration data
;* There are two parameters to configure, CARD_BASE_PORT and
;* NUMBER_OF_RACKS. All 7507 and 7508 cards must be mapped contiguously
;* beginning at the CARD_BASE_PORT, all 7507 cards occupy 4 ports
;* and 7508 cards occupy 8 port addresses.
;* The NUMBER_OF_RACKS parameter may be determined by the
;* number of 7507 and 7508 cards. The 7507 drives 1 rack and the
;* 7508 drives either 1 or 2 racks.
;* This routine resets all racks to a known state, power off
;* and all rack outputs set low.
;*
;* inputs: NUMBER_OF_RACKS = 8H
;*          7508_BASE_PORT = 0X
;*
;* output: AX = 0 ;base port address and number of racks are initialized*
;*
;*****

C INCLUDE      STD.LIB.INC
C              .list
C

      public    @NUMBER_OF_RACKS,@7508_BASE_PORT ; public parameters
      public    @ACUR_7508_DATA                ;

      DATASTART                                ; create static data area
1 DATA      SEGMENT WORD PUBLIC 'DATA'
      @NUMBER_OF_RACKS DW 2                      ; number of racks default is 2
      @7508_BASE_PORT DW 0150H                  ; Default port address
      @ACUR_7508_DATA DB 88 DUP (00h)           ; Current port data

0000
0000 0002
0002 0150
0004 0058(
00
1

      DATAEND
005C 1 DATA      ENDS

      MODREG      <DX,BX,ES>
      START       pl7508_init,<DX,BH>,0
1               .lall
0000 1 pl7508_TEXT SEGMENT BYTE PUBLIC 'CODE'
1               ASSUME CS:pl7508_TEXT,DS:NOTHING,ES:NOTHING,SS:NOTHING
1               PUBLIC pl7508_init
0000 1 @0500 dw ? ;save area for data segment
0002 1 pl7508_init PROC FAR
0002 4 PUSH ES
0003 4 PUSH DX
0004 4 PUSH BX
0005 2E: 8C 1E 0000 R 1 MOV @0500, DS ;save data segment

      DATASEG      DS
000A 50 1 PUSH AX
000B 88 ---- R 1 MOV AX,SEG DATA
000E 8E DB 1 MOV DS,AX
0010 58 1 POP AX
0011 32 DB xor bl,bl
0013 86 DF xchg bl,bh
0015 89 1E 0000 R mov @NUMBER_OF_RACKS,BX ; set number of racks
0019 89 16 0002 R mov @7508_BASE_PORT,dx ; save new value for base port address
001D 88 0E 0000 R mov cx,[@NUMBER_OF_RACKS] ; number of ports to reset
; reset all 750x racks
0021 8D 1E 0004 R lea bx,@ACUR_7508_DATA ; get address of port data
0025 80 00 mov al,0
0027 88 07 mov [bx],al ; move data into current port data

```

EXAMPLE ASSEMBLY LANGUAGE OBJECT CODE LISTING (Continued)

Microsoft (R) Macro Assembler Version 4.00

5/19/86 13:45:33

Page 1-2

```

Function: pl7508_init      Library: Assembly language

0029 43                      inc bx                ; increment bx to next port data
002A EE                      out dx,al              ; send data to port
002B 42                      inc dx                ; increment port address
002C 88 07                   mov [bx],al          ; move 00 to 2nd port of rack
002E 43                      inc bx                ;
002F EE                      out dx,al              ;
0030 42                      inc dx                ;
0031 88 07                   mov [bx],al          ; mov 00 to 3rd port of rack
0033 43                      inc bx                ;
0034 EE                      out dx,al              ;
0035 42                      inc dx                ;
0036 FE C8                   dec al              ; send ff to rack power port
0038 88 07                   mov [bx],al          ;
003A 43                      inc bx                ;
003B EE                      out dx,al              ;
003C 42                      inc dx                ;
003D E2 E6                   loop mov_loop          ;
003F 33 C0                   xor ax,ax          ; clear return value

0041                      exit:
                        FINISH      pl7508_init      ; restore registers and return
0041 2E: 8E 1E 0000 R      1      MOV      DS, @a05a2a ; restore data segment
0046 5B                      4      POP      BX
0047 5A                      4      POP      DX
0048 07                      4      POP      ES
0049 C8                      1      RET
                        1      .lall
                        1 pl7508_init      ENDP
004A                      1 pl7508_TEXT   ENDS
                        END

```

RETURNED VALUE

All library routines return at least one value. More values can be returned if the parameters passed include pointers to receiving fields in the calling program. This is described in the preceding Parameter Passing subsection. Since this procedure for returning more than one value is somewhat cumbersome (and slightly dangerous since pointer errors are easy to make and hard to discover), it is generally better to try to map multiple results into the single required return value. If normal return values are always within a special range, then values outside that range may be used to indicate errors. For example, if all the normal returned values are positive, error conditions could be returned as negative numbers.

GENERAL PRACTICES

- If the module source code modifies the direction flag (e.g., uses a block move), it must be cleared (CLD) before the routine is exited.
- No absolute addresses should be used in the module source code.
- Any symbols declared as "public" required by the module for internal use (that are not to be called by an application program) should have a name preceded by @@. This prevents the name from being accidentally referenced later.
- The routines should be written to support more than one STD card if it would be reasonable to use more than one card in a system. The routines should be written to support up to 22 cards of the same type.
- A parameter is always returned from a routine. If there is no returned value required, the AX register should be set to 0 before return. This prevents random values from being returned.

- Each STD card supported by the library should have an initialize routine (INIT), a reset routine (RESET), and a self-test routine (TEST). These routines should exist even if they don't affect the STD card. This allows generalized programming for initialize, reset, and self-test functions.
- When applicable, the initialize routine should not have to be called when only one card of each type is used in the system, configured in its as-shipped condition. This is accomplished by initializing the as-shipped condition in the DATASTART storage area. These RAM locations are overwritten with the parameters passed to the INIT routine if it is called.
- Always use the Load Effective Address instruction (LEA) to obtain the address of an operand. Using "MOV Y, OFFSET X" to get the location of X can cause a problem with the assembler in some cases. This occurs because the OFFSET operator takes the segment-relative offset, when the requirement of the code (due to the macros) may be a group-relative offset.
- The code for a library module must not exceed 64K, since it must fit within one code segment. For small model C application programs, the application code and the library modules must all fit within 64K.

DOCUMENTING NEW MODULES

New library modules can be documented similarly to those already in the library. This can be accomplished by using a similar format and a word processor or by typing on photocopies of the following blank forms.

To allow library modules for several cards to be added to the library, the page numbering convention of Sections 4, 5, and 6 should be used. Place all library routines for a given language in alphabetical order and number in the form: (section number)-(card model number)-(page number). This scheme allows new modules to be added anywhere in Sections 4, 5, and 6 without disturbing the existing page numbering.

If you are using a word processor for documentation, you should use margins to cause the text to fit onto the same size page as this manual is

printed on. The processed pages can then be cut to fit into the manual. For Microsoft WORD, the following Format Division setup works well:

page length: 11", width: 8.5",
gutter width: 0", margin top 0.5",
bottom: 3.7", left: 2.0", right: 2.25"
running head pos from bottom: 3.2"

This setup spaces the text equally from the left and right margins. This allows the text to be photocopied on both sides of a page and then to be cut to fit into the manual binder.

Structure:

Description:

Input
Parameters:

Returned
Value:

Example:

Structure:

Description:

Input
Parameters:

Returned
Value:

Example:

Structure:

Description:

Input
Parameters:

Returned
Value:

Example:





Section 4

C LANGUAGE LIBRARY

CONTENTS	PAGE
Introduction	4-1
Analog Devices RTI-1225 Analog Input/Output Card	4-1225-1
ad1225_ain	4-1225-4
ad1225_aout	4-1225-5
ad1225_init	4-1225-7
ad1225_reset	4-1225-11
ad1225_test	4-1225-12
Analog Devices RTI-1226 Analog Input Card	4-1226-1
ad1226_ain	4-1226-4
ad1226_init	4-1226-5
ad1226_reset	4-1226-8
ad1226_test	4-1226-9
Analog Devices RTI-1260 Analog Input Card	4-1260-1
ad1260_ain	4-1260-4
ad1260_init	4-1260-5
ad1260_reset	4-1260-8
ad1260_test	4-1260-9
Analog Devices RTI-1262 Analog Output Card	4-1262-1
ad1262_amps	4-1262-4
ad1262_aout	4-1262-6
ad1262_init	4-1262-8
ad1262_reset	4-1262-11

Pro-Log 75XX, 76XX, and 79XX Input/Output Cards

4-7000-1

pl7000_init	4-7000-3
pl7000_rdbit	4-7000-5
pl7000_rdport	4-7000-6
pl7000_resetport	4-7000-7
pl7000_wrtbit	4-7000-8
pl7000_wrtbitrd	4-7000-9
pl7000_wrtport	4-7000-11
pl7000_wrtportrd	4-7000-12
7501 DC Driver Card	4-7000-13
7502 SPST Relay Output Card	4-7000-13
7503 Optoisolated Input Card	4-7000-14
7504 Triac Output Card	4-7000-14
7506 Optoisolated Input Card	4-7000-15
7601 TTL Input/Output Port Card	4-7000-16
7602 TTL Output Port Card	4-7000-17
7603A TTL Input Port Card	4-7000-18
7604A TTL Input/Output Card	4-7000-19
7605 Programmable TTL I/O Card	4-7000-20
7614 64-Line TTL I/O Card	4-7000-21
7904 Decoded I/O Utility Card	4-7000-22
7914 Decoded I/O Utility Card	4-7000-23

Pro-Log 7303 Keyboard Display Card

4-7303-1

pl7303_chrshft	4-7303-3
pl7303_chrrwr	4-7303-4
pl7303_cursor	4-7303-5
pl7303_getkey	4-7303-6
pl7303_getrockr	4-7303-7
pl7303_init	4-7303-8
pl7303_keyfetch	4-7303-10
pl7303_keystat	4-7303-11
pl7303_message	4-7303-12
pl7303_rckrstat	4-7303-13
pl7303_reset	4-7303-14
pl7303_setleds	4-7303-15
pl7303_test	4-7303-16
pl7303_wait	4-7303-17
pl7303_word	4-7303-18

**Pro-Log 7312/7314A
Communications Card**

4-7314-1

pl7314__bufflush	4-7314-3
pl7314__chanrst	4-7314-4
pl7314__init	4-7314-6
pl7314__inten	4-7314-8
pl7314__rcv	4-7314-10
pl7314__rcvchk	4-7314-11
pl7314__setbuf	4-7314-12
pl7314__test	4-7314-14
pl7314__xmt	4-7314-15
pl7314__xmtbrk	4-7314-16
pl7314__xmtchk	4-7314-17
pl7314__xmtimd	4-7314-18

**Pro-Log 7507/7508
TTL Input/Output Card**

4-7508-1

pl7508__init	4-7508-2
pl7508__rackpwr	4-7508-4
pl7508__readmod	4-7508-5
pl7508__reset	4-7508-6
pl7508__setmod	4-7508-7
pl7508__test	4-7508-8

**Pro-Log 7700 Memory Diagnostic
Routines**

4-7700-1

pl7700__checksum	4-7700-2
pl7700__memtest	4-7700-2

Pro-Log 7863 CPU Card

4-7863-1

pl7863_chkmark	4-7863-3
pl7863_getdate	4-7863-4
pl7863_gettime	4-7863-5
pl7863_getwatch	4-7863-6
pl7863_init	4-7863-7
pl7863_setalarm	4-7863-9
pl7863_setmark	4-7863-11
pl7863_test	4-7863-13
pl7863_wait	4-7863-14
pl7863_weekday	4-7863-15

Pro-Log 7888 CPU Card

4-7888-1

pl7888_chkmark	4-7888-3
pl7888_getdate	4-7888-4
pl7888_gettime	4-7888-5
pl7888_getwatch	4-7888-6
pl7888_init	4-7888-7
pl7888_setalarm	4-7888-9
pl7888_setmark	4-7888-11
pl7888_test	4-7888-13
pl7888_wait	4-7888-14
pl7888_weekday	4-7888-15

C LANGUAGE LIBRARY

INTRODUCTION

This section contains the STD-LIB routines for application programs written in Microsoft C.

Three memory models for Microsoft C are supported by the library: the S (small) model, the M (medium) model, and the L (large) model. Your application program should be linked with the STD-LIB version corresponding to the model used in the application program. The versions of STD-LIB are shown in the following table:

STD-LIB VERSION	MODEL	PROGRAM SEG. SIZE	DATA SEG. SIZE
STDLIBCS.LIB	S	≤ 64K bytes	≤ 64K
STDLIBCM.LIB	M	> 64K bytes	≤ 64K
STDLIBCL.LIB	L	> 64K bytes	> 64K

Refer to Section 2, Operation and Programming, for a description of how to link the appropriate version of STD-LIB with your application program.

The Analog Devices RTI-1225 Analog Input/Output Card is a general-purpose, low-cost analog I/O card. The RTI-1225 card uses an analog multiplexer to select among analog input channels that are connected to a 10-bit A/D converter. The input channels can be configured to be 16 single-ended or 8 differential inputs. There are two 8-bit D/A converters providing two output channels.

The STD-LIB RTI-1225 software modules are as follows:

- ad1225_ain:** Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest millivolt (based on the default hardware configuration or the configuration set by `ad1225_init`).
- ad1225_aout:** Transforms the output voltage requested in the call from millivolts to the nearest D/A code and writes an output voltage.
- ad1225_init:** Initializes the library software with the user-selected hardware configuration of the card. If this routine has been invoked properly, a call is made to the `ad1225_reset` routine.
- ad1225_reset:** Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout. The D/A input registers are set to output a voltage of 0V.

ad1225_test: Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.

NOTE: The RTI-1225 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal B0 which results in the card using ports BB through BF.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1225 card that has been remapped to the library default would be defined as a base address of 1B0 in the initialize command, but the physical ports of the card are actually located at 1BB through 1BF. When referring to contiguous mapping for RTI-1225 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy five ports per card.

To change an RTI-1225 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 59-60, 62-63, 51-52
install 61-63, 58-59, 53-55, 15-16

For technical questions concerning STD-LIB software for the Analog Devices RTI-1225, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1225 hardware, contact the appropriate Analog Devices customer service as follows:

STATE	PHONE NUMBER
Alaska	
Arizona	(714)-641-9391
California (southern)	Direct Sales Office
Colorado	
California (northern)	(408)-947-0633
Utah	Direct Sales Office
Delaware	(215)-643-7790
New Jersey	Direct Sales Office
Pennsylvania	
Maine	
Massachusetts	(617)-329-4700
New Hampshire	Factory
Rhode Island	
Illinois	
Indiana	
Iowa	
Kansas	
Kentucky	(312)-653-5000
Missouri	Direct Sales Office
North Dakota	
South Dakota	
Wisconsin	
All Other States	(617)-329-4700
	Factory

Structure:	<pre>extern int ad1225_ain(); int cardno; int channel; ad1225_ain(cardno,channel);</pre>
Description:	This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in millivolts based on the user-selected input configuration.
Input Parameters:	cardno is an integer value specifying the card from which the reading is to be taken (0 - 21). channel is an integer value specifying the input channel to be read (0 - 15).
Returned Value:	ad1225_ain contains the analog voltage read in millivolts (-10000 to + 9990) or contains an error code: -32768 = channel number out of range -32767 = board number out of range -32766 = timeout error during conversion
Example:	The following C program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system. <pre>voltage=ad1225_ain(3,2);</pre>

RTI-1225

C library

Structure:	<pre>extern int ad1225_aout(); int cardno; int channel; int voltage; ad1225_aout(cardno,channel,voltage);</pre>
Description:	This routine outputs a voltage to the selected channel. The selected voltage is input in millivolts and converted to a binary program value for the D/A converter based on the user-selected output configuration for the channel.
Input Parameters:	cardno is an integer value specifying the card from which the voltage is to be output (0 - 21). channel is an integer value specifying the channel from which the voltage is to be output (0 or 1). voltage contains the analog voltage to be output in millivolts (-10000 to + 9961).
Returned Value:	ad1225_aout contains the returned integer error code: 0 = no error -1 = channel number out of range -2 = board number out of range -3 = voltage parameter too large (clipped to maximum value) -4 = voltage parameter too small (clipped to minimum value)
Example:	The following C program segment could be written to output an analog voltage of 2V from

ad1225__aout

output channel 1 of the fourth card in the system.

errcode=ad1225__aout(3,1,2000);

RTI-1225

C library

-
- Structure:**
- ```
extern int ad1225_init();
int baseno;
int baseadr;
int nocards;
int inchan;
int adconf;
int da0conf;
int da1conf;
ad1225_init(baseno,baseadr,nocards,
inchan,adconf,da0conf,da1conf);
```
- Description:**
- This routine initializes the RTI-1225 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1225 card in a system with a base address of 1B0, configured for 16 single-ended inputs of  $\pm 10V$  two's complement and both outputs  $\pm 10V$  and two's complement. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call. If there are no parameter errors, the card is software reset with a call to the ad1225\_reset routine.
- Input Parameters:**
- baseno** is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).
- baseadr** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1225 cards. The library routines use a 10-bit address. The RTI-1225 card should

be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1225 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

**nocards** is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

**inchan** is the input multiplexer configuration:

- 01 = single-ended
- 02 = differential
- 03 = pseudo-differential

**adconf** is the A/D input range and output code:

- 00 = 0V to +10V, straight binary
- 01 = -5V to +5V, two's complement
- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, two's complement
- 04 = -10V to +10V, offset binary

**da0conf** is the D/A channel 0 output range and input code:

- 00 = 0V to +10V, straight binary
- 01 = -5V to +5V, two's complement
- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, two's complement
- 04 = -10V to +10V, offset binary

**da1conf** is the D/A channel 1 output range and input code:

- 00 = 0V to +10V, straight binary
- 01 = -5V to +5V, two's complement

- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, two's complement
- 04 = -10V to +10V, offset binary

Returned  
Value:

**ad1225\_init** contains the returned  
integer error code:

- 0 = no error
- 1 = board number out of range
- 2 = invalid card count  
(nocards >22 or the number of  
cards causes the base address  
of the last card to be less than  
that of the first card (wrap  
around))
- 3 = invalid multiplexer  
configuration
- 4 = invalid A/D input range/output  
code
- 5 = invalid D/A channel 0 input  
code/output range
- 6 = invalid D/A channel 1 input  
code/output range
- 7 = timeout error during reset

Example:

The following C program segment could be written to initialize the library for a group of four RTI-1225 cards after a group of three differently configured RTI-1225 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 0x20. The cards are configured for differential inputs, 0 to +10V input range with straight binary code, and both output channels for 0 to +10V output range with straight binary code.

```
errcode=ad1225_init(3,0x120,4,02,00,00,00);
```

---

Structure:            `extern int ad1225_reset();`  
                      `ad1225_reset();`

Description:            This routine software resets all the RTI-1225 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed. The D/A input registers are set to output a voltage of 0V. Note that the D/A output voltage may differ between hardware and software resets, depending on the user-selected hardware configuration.

Input  
Parameters:            None

Returned  
Value:                 `ad1225_reset` contains the returned integer error code:

                      0 = no error

                      -7 = timeout error during reset

Example:                The following C program segment resets all RTI-1225 cards in a system.

```
errcode=ad1225_reset();
```

RTI-1225

C library

Structure:           extern int ad1225\_test();  
                      int cardno;  
                      ad1225\_test(cardno);

Description:        This routine self-tests an RTI-1225 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.

Input Parameters:    **cardno** is an integer value specifying the card to be tested (0 - 21).

Returned Value:     **ad1225\_test** contains the returned integer error code:

0 = no error  
-1 = invalid card selected  
-7 = timeout error

Example:            The following C program segment tests the second RTI-1225 card in a system.

```
errcode=ad1225_test(1);
```



**Analog Devices**  
**RTI-1226**  
**Analog Input Card**  
C library

---

AD1226

The Analog Devices RTI-1226 Analog Input Card is a general-purpose, low-cost analog I/O card. The RTI-1226 card uses an analog multiplexer to select among analog input channels that are connected to a 10-bit A/D converter. The input channels can be configured to be 16 single-ended or 8 differential inputs.

The STD-LIB RTI-1226 software modules are as follows:

- |                      |                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ad1226_ain:</b>   | Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest millivolt (based on the default hardware configuration or the configuration set by <code>ad1226_init</code> ). |
| <b>ad1226_init:</b>  | Initializes the library software with the user-selected hardware configuration of the card. If this routine has been invoked properly, a call is made to the <code>ad1226_reset</code> routine.                       |
| <b>ad1226_reset:</b> | Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout.                                            |
| <b>ad1226_test:</b>  | Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.                                                                                        |

**NOTE:** The RTI-1226 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal B0 which results in the card using ports BB through BD.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1226 card that has been remapped to the library default would be defined as a base address of 1B0 in the initialize command, but the physical ports of the card are actually located at 1BB through 1BD. When referring to contiguous mapping for RTI-1226 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy three ports per card.

To change an RTI-1226 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 59-60, 62-63, 51-52  
install 61-63, 58-59, 53-55, 15-16

For technical questions concerning STD-LIB software for the Analog Devices RTI-1226, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1226 hardware, contact the appropriate Analog Devices customer service as follows:

| STATE                 | PHONE NUMBER              |
|-----------------------|---------------------------|
| Alaska                |                           |
| Arizona               | (714)-641-9391            |
| California (southern) | Direct Sales Office       |
| Colorado              |                           |
| California (northern) | (408)-947-0633            |
| Utah                  | Direct Sales Office       |
| Delaware              | (215)-643-7790            |
| New Jersey            | Direct Sales Office       |
| Pennsylvania          |                           |
| Maine                 |                           |
| Massachusetts         | (617)-329-4700            |
| New Hampshire         | Factory                   |
| Rhode Island          |                           |
| Illinois              |                           |
| Indiana               |                           |
| Iowa                  |                           |
| Kansas                |                           |
| Kentucky              | (312)-653-5000            |
| Missouri              | Direct Sales Office       |
| North Dakota          |                           |
| South Dakota          |                           |
| Wisconsin             |                           |
| All Other States      | (617)-329-4700<br>Factory |

---

|                   |                                                                                                                                                                                                                                                                                                        |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int ad1226_ain();<br/>int cardno;<br/>int channel;<br/>ad1226_ain(cardno,channel);</pre>                                                                                                                                                                                                   |
| Description:      | This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in millivolts based on the user-selected input configuration.                                                                                                 |
| Input Parameters: | <p><b>cardno</b> is an integer value specifying the card from which the reading is to be taken (0 - 21).</p> <p><b>channel</b> is an integer value specifying the input channel to be read (0 - 15).</p>                                                                                               |
| Returned Value:   | <p><b>ad1226_ain</b> contains the analog voltage read in millivolts (-10000 to + 9990) or contains an error code:</p> <ul style="list-style-type: none"><li>-32768 = channel number out of range</li><li>-32767 = board number out of range</li><li>-32766 = timeout error during conversion</li></ul> |
| Example:          | <p>The following C program segment could be written to read an analog voltage from input channel 2 of the fourth card in the system.</p> <pre>voltage=ad1226_ain(3,2);</pre>                                                                                                                           |

Structure:           extern int ad1226\_init();  
                      int baseno;  
                      int baseadr;  
                      int nocards;  
                      int inchan;  
                      int adconf;  
                      ad1226\_init(baseno,baseadr,nocards,  
                      inchan,adconf);

Description:           This routine initializes the RTI-1226 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1226 card in a system with a base address of 1B0, configured for 16 single-ended inputs of  $\pm 10V$  two's complement. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call. If there are no parameter errors, the card is software reset with a call to the ad1226\_reset routine.

Input Parameters:       **baseno** is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).

**baseadr** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1226 cards. The library routines use a 10-bit address. The RTI-1226 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1226

## ad1226\_init

address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

**nocards** is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

**inchan** is the input multiplexer configuration:

- 01 = single-ended
- 02 = differential
- 03 = pseudo-differential

**adconf** is the A/D input range and output code:

- 00 = 0V to +10V, straight binary
- 01 = -5V to +5V, two's complement
- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, two's complement
- 04 = -10V to +10V, offset binary

Returned  
Value:

**ad1226\_init** contains the returned integer error code:

- 0 = no error
- 1 = board number out of range
- 2 = invalid card count  
(nocards > 22 or the number of cards causes the base address of the last card to be less than that of the first card (wrap around))
- 3 = invalid multiplexer configuration
- 4 = invalid A/D input range/output code
- 7 = timeout error during reset

## Example:

The following C program segment could be written to initialize the library for a group of four RTI-1226 cards after a group of three differently configured RTI-1226 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 0x20. The cards are configured for differential inputs, 0 to +10V input range with straight binary code.

```
errcode=ad1226_init(3,0x120,4,02,00);
```

---

Structure:           extern int ad1226\_reset();  
                      ad1226\_reset();

Description:        This routine software resets all the RTI-1226 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed.

Input  
Parameters:        None

Returned  
Value:             **ad1226\_reset** contains the returned integer error code:

                    0 = no error  
                    -7 = timeout error during reset

Example:           The following C program segment resets all RTI-1226 cards in a system.

```
errcode=ad1226_reset();
```

---

|                   |                                                                                                                                          |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int ad1226_test();<br/>int cardno;<br/>ad1226_test(cardno);</pre>                                                            |
| Description:      | This routine self-tests an RTI-1226 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error. |
| Input Parameters: | <b>cardno</b> is an integer value specifying the card to be tested (0 - 21).                                                             |
| Returned Value:   | <b>ad1226_test</b> contains the returned integer error code:<br><br>0 = no error<br>-1 = invalid card selected<br>-7 = timeout error     |
| Example:          | The following C program segment tests the second RTI-1226 card in a system.<br><br><pre>errcode=ad1226_test(1);</pre>                    |



The Analog Devices RTI-1260 Analog Input Card is a general-purpose, low-cost analog I/O card. The RTI-1260 card uses an analog multiplexer to select among analog input channels that are connected to a 12-bit A/D converter. The input channels can be configured to be 32 single-ended or 16 differential inputs.

The STD-LIB RTI-1260 software modules are as follows:

- |                      |                                                                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ad1260_ain:</b>   | Reads an input voltage and converts the A/D converter binary value to a reading rounded to the nearest microvolt (based on the default hardware configuration or the configuration set by <code>ad1260_init</code> ). |
| <b>ad1260_init:</b>  | Initializes the library software with the user-selected hardware configuration of the card.                                                                                                                           |
| <b>ad1260_reset:</b> | Performs a software reset of the card. The multiplexer is set to select input channel 0 and wait for any pending conversions to complete or a conversion busy bit timeout.                                            |
| <b>ad1260_test:</b>  | Performs a self-test on one card. The routine reads a voltage from the card and passes self-test if there is no timeout error.                                                                                        |

**NOTE:** The RTI-1260 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal 30 which results in the card using ports 3B through 3D.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1260 card that has been remapped to the library default would be defined as a base address of 130 in the initialize command, but the physical ports of the card are actually located at 13B through 13D. When referring to contiguous mapping for RTI-1260 cards, the cards are mapped as though they occupied 16 ports per card, even though they only physically occupy three ports per card.

To change an RTI-1260 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 48-50, 51-52, 44-45  
install 49-50, 52-53, 45-47, 15-16, 7-8

For technical questions concerning STD-LIB software for the Analog Devices RTI-1260, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1260 hardware, contact the appropriate Analog Devices customer service as follows:

| STATE                 | PHONE NUMBER              |
|-----------------------|---------------------------|
| Alaska                |                           |
| Arizona               | (714)-641-9391            |
| California (southern) | Direct Sales Office       |
| Colorado              |                           |
| California (northern) | (408)-947-0633            |
| Utah                  | Direct Sales Office       |
| Delaware              | (215)-643-7790            |
| New Jersey            | Direct Sales Office       |
| Pennsylvania          |                           |
| Maine                 |                           |
| Massachusetts         | (617)-329-4700            |
| New Hampshire         | Factory                   |
| Rhode Island          |                           |
| Illinois              |                           |
| Indiana               |                           |
| Iowa                  |                           |
| Kansas                |                           |
| Kentucky              | (312)-653-5000            |
| Missouri              | Direct Sales Office       |
| North Dakota          |                           |
| South Dakota          |                           |
| Wisconsin             |                           |
| All Other States      | (617)-329-4700<br>Factory |

---

Structure:           extern long ad1260\_\_ain();  
                      int cardno;  
                      int channel;  
                      ad1260\_\_ain(cardno,channel);

Description:       This routine reads an input voltage from the selected channel. The binary reading of the A/D converter is translated to a binary reading in microvolts based on the user-selected input configuration.

Input               **cardno** is an integer value specifying the card  
Parameters:       from which the reading is to be taken (0 - 21).  
  
                      **channel** is an integer value specifying the input  
                      channel to be read (0 - 31).

Returned           **ad1260\_\_ain** contains the analog voltage  
Value:           read in microvolts or contains an error code:

-20,000,000 = channel number out of range  
-30,000,000 = board number out of range  
-40,000,000 = timeout error during conversion

Example:           The following C program segment could be  
                      written to read an analog voltage from input  
                      channel 2 of the fourth card in the system.

voltage=ad1260\_\_ain(3,2);

---

Structure:           extern int ad1260\_init();  
                      int baseno;  
                      int baseadr;  
                      int nocards;  
                      int inchan;  
                      long gain;  
                      int adconf;  
                      **ad1260\_init(baseno,baseadr,nocards,**  
                      **inchan,gain,adconf);**

Description:           This routine initializes the RTI-1260 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1260 card in a system with a base address of 130, gain equal one, configured for 32 single-ended inputs of  $\pm 10V$  offset binary. There may be up to 22 of these cards in a system. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call.

Input Parameters:       **baseno** is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).

**baseadr** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1260 cards. The library routines use a 10-bit address. The RTI-1260 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1260 address 20 is entered as 120). The least

## ad1260\_init

significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

**nocards** is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

**inchan** is the input multiplexer configuration:

- 00 = single-ended
- 01 = differential
- 02 = pseudo-differential

**gain** is the voltage gain on the A/D input multiplied by a factor of 10,000 to enable 12-bits of accuracy, hence a gain 1.2346 is entered as 12,346. The allowable gain range is 10000 - 10000000, this corresponds to a gain of 1V/V to 1000V/V.

**adconf** is the A/D input range and output code:

- 00 = 0V to +10V, straight binary
- 01 = -10V to +10V, offset binary
- 02 = -10V to +10V, two's complement

Returned  
Value:

**ad1260\_init** contains the returned  
integer error code:

- 0 = no error
- 1 = board number out of range
- 2 = invalid card count (nocards >22)
- 3 = invalid multiplexer configuration
- 4 = invalid A/D input range/output code
- 5 = timeout error during reset
- 6 = gain out of range

Example:

The following C program segment could be written to initialize the library for a group of

four RTI-1260 cards after a group of three differently configured RTI-1260 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 0x20. The cards are configured for differential inputs, gain equal 1V/V, 0 to +10V input range with straight binary code.

```
gain = 10000;
errcode=ad1260_init(3,0x120,4,01,gain,00);
```

---

Structure:               extern int ad1260\_\_reset();  
                          ad1260\_\_reset();

Description:            This routine software resets board 0 and all initialized RTI-1260 cards in the system. The multiplexer is set to select input channel 0, a wait for any pending conversions to complete or a timeout is performed.

Input  
Parameters:            None

Returned  
Value:                 **ad1260\_\_reset** contains the returned integer error code:

                          0 = no error  
                          -5 = timeout error during reset

Example:               The following C program segment resets all RTI-1260 cards in a system.

```
errcode=ad1260__reset();
```

RTI-1260

C library

---

Structure:           extern int ad1260\_test();  
                      int cardno;  
                      ad1260\_test(cardno);

Description:        This routine self-tests an RTI-1260 card. A voltage is read from the A/D converter and self-test is passed if there is no timeout error.

Input  
Parameters:         **cardno** is an integer value specifying the card to be tested (0 - 21).

Returned  
Value:             **ad1260\_test** contains the returned integer error code:

                    0 = no error  
                    -1 = invalid card selected  
                    -5 = timeout error

Example:            The following C program segment tests the second RTI-1260 card in a system.

```
errcode=ad1260_test(1);
```



**Analog Devices**  
**RTI-1262**  
**Analog Output Card**  
C library

---

The Analog Devices RTI-1262 Analog Output Card is a general-purpose, low-cost analog output card. There are four 12-bit D/A converters providing four output channels. Two of the four voltage outputs can be converted to a 4-20 milliamp current loop output by installing the correct hardware option.

The STD-LIB RTI-1262 software modules are as follows:

|                      |                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ad1262_amps</b>   | Transforms the output current requested in the call from microamps to the nearest D/A code and writes an output current. The output current is in the range 4000 - 20000 microamps. |
| <b>ad1262_aout:</b>  | Transforms the output voltage requested in the call from millivolts to the nearest D/A code and writes an output voltage.                                                           |
| <b>ad1262_init:</b>  | Initializes the library software with the user-selected hardware configuration of the card.                                                                                         |
| <b>ad1262_reset:</b> | Performs a software reset of the card. The D/A input registers are set to output a voltage of 0V.                                                                                   |

**NOTE:** The RTI-1262 as-shipped mapping is at hexadecimal base address FFF0 in the memory map. To be compatible with the library, the card needs to be port mapped with an 8-bit address. The default 8-bit base address for the library is hexadecimal C0 which results in the card using ports C0 through C7.

The Analog Devices convention is to define the base address of a card as an address with the last four bits being zero. The library has followed this convention. This means that an RTI-1262 card that has been remapped to the library default would be defined as a base address of 1C0 in the initialize command, the physical ports of the card are actually located at 1C0 through 1C7. When referring to contiguous mapping for RTI-1262 cards, the cards are mapped as though they occupied 8 ports per card.

To change an RTI-1262 card from its as-shipped mapping to the library default mapping, make the following changes to the jumpers:

remove 58-60, 55-56, 51-52  
install 59-60, 56-57, 52-53, 9-10, 1-2

For technical questions concerning STD-LIB software for the Analog Devices RTI-1262, contact Pro-Log customer service. For technical questions concerning the operation of the Analog Devices RTI-1262 hardware, contact the appropriate Analog Devices customer service as follows:

| STATE                 | PHONE NUMBER              |
|-----------------------|---------------------------|
| Alaska                |                           |
| Arizona               | (714)-641-9391            |
| California (southern) | Direct Sales Office       |
| Colorado              |                           |
| California (northern) | (408)-947-0633            |
| Utah                  | Direct Sales Office       |
| Delaware              | (215)-643-7790            |
| New Jersey            | Direct Sales Office       |
| Pennsylvania          |                           |
| Maine                 |                           |
| Massachusetts         | (617)-329-4700            |
| New Hampshire         | Factory                   |
| Rhode Island          |                           |
| Illinois              |                           |
| Indiana               |                           |
| Iowa                  |                           |
| Kansas                |                           |
| Kentucky              | (312)-653-5000            |
| Missouri              | Direct Sales Office       |
| North Dakota          |                           |
| South Dakota          |                           |
| Wisconsin             |                           |
| All Other States      | (617)-329-4700<br>Factory |

AD1262

Structure:           extern int ad1262\_amps();  
                      int cardno;  
                      int channel;  
                      int amps;  
                      ad1262\_amps(cardno,channel,amps);

Description:           This routine outputs a current to the selected channel. The selected current is input in microamps and converted to a binary program value for the V/I converter. The V/I converter must be installed for the channel and the channel must be configured for range 0 - 10 volts, straight binary encoding.

Input Parameters:       **cardno** is an integer value specifying the card from which the current is to be output (0 - 21).  
  
                          **channel** is an integer value specifying the output channel (0 - 3).  
  
                          **amps** is an integer value to select the current to be output in microamps (4000 - 20000).

Returned Value:        **ad1262\_amps** contains the error code:

- 0 = no error
- 1 = board number out of range
- 2 = channel number out of range
- 3 = current out of range

Example:

The following C program segment could be written to output a current of 5000 microamps to channel 0 of the fourth card in the system.

```
errcode=ad1262_amps(3,0,5000)
```

---

|                   |                                                                                                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int ad1262_aout();<br/>int cardno;<br/>int channel;<br/>int voltage;<br/>ad1262_aout(cardno,channel,voltage);</pre>                                                                                                                                                                                                 |
| Description:      | This routine outputs a voltage to the selected channel. The selected voltage is input in millivolts and converted to a binary program value for the D/A converter based on the user-selected output configuration for the channel.                                                                                              |
| Input Parameters: | <p><b>cardno</b> is an integer value specifying the card from which the voltage is to be output (0 - 21).</p> <p><b>channel</b> is an integer value specifying the channel from which the voltage is to be output (0 - 3).</p> <p><b>voltage</b> contains the analog voltage to be output in millivolts (-10000 to + 9990).</p> |
| Returned Value:   | <p><b>ad1262_aout</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = channel number out of range</li><li>-2 = board number out of range</li><li>-3 = voltage parameter out of range</li></ul>                                                                    |

Example:

The following C program segment could be written to output an analog voltage of 2V from output channel 1 of the fourth card in the system.

```
errcode=ad1262_aout(3,1,2000);
```

Structure:

```
extern int ad1262__init();
int baseno;
int baseadr;
int nocards;
int da0conf;
int da1conf;
int da2conf;
int da3conf;
ad1262__init(baseno,baseadr,nocards,
da0conf,da1conf,da2conf,da3conf);
```

Description:

This routine initializes the RTI-1262 library routines with the user-selected hardware configuration of the card if different from the default condition. The card default initialization is for one RTI-1262 card in a system with a base address of 1C0, configured for -10V to +10V offset binary encoding.

Input Parameters:

**baseno** is an integer value specifying the base number of the group of contiguous cards configured by this command (0 - 21).

**baseadr** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped RTI-1262 cards. The library routines use a 10-bit address. The RTI-1262 card should be mapped for an 8-bit I/O address. In software, set the most significant bits of the 10-bit address to 01 (for example, the RTI-1262 address 20 is entered as 120). The least significant four bits of the address must be 0. The routines do not work if the card is memory mapped.

**nocards** is an integer value specifying the number of contiguously mapped cards specified by this command (1 - 22).

**da0conf** is the D/A channel 0 output range and input code:

- 00 = 0V to +5V, straight binary
- 01 = 0V to +10V, straight binary
- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, offset binary

**da1conf** is the D/A channel 1 output range and input code:

- 00 = 0V to +5V, straight binary
- 01 = 0V to +10V, straight binary
- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, offset binary

**da2conf** is the D/A channel 2 output range and input code:

- 00 = 0V to +5V, straight binary
- 01 = 0V to +10V, straight binary
- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, offset binary

**da3conf** is the D/A channel 3 output range and input code:

- 00 = 0V to +5V, straight binary
- 01 = 0V to +10V, straight binary
- 02 = -5V to +5V, offset binary
- 03 = -10V to +10V, offset binary

## ad1262\_\_init

Returned  
Value:

**ad1262\_\_init** contains the returned  
integer error code:

- 0 = no error
- 1 = board number out of range
- 2 = invalid card count
- 3 = invalid D/A channel input  
code/output range

Example:

The following C program segment could be written to initialize the library for a group of four RTI-1262 cards after a group of three differently configured RTI-1262 cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 0x20. Output channels are configured for 0 to +10V output range with straight binary code.

```
errcode=ad1262__init(3,0x120,4,01,01,01,01);
```

Structure:           extern int ad1262\_reset();  
                      ad1262\_reset();

Description:        This routine software resets all the RTI-1262 cards in the system. The D/A input registers are set to output a voltage of 0V. Note that the D/A output voltage may differ between hardware and software resets, depending on the user-selected hardware configuration.

Input  
Parameters:         None

Returned  
Value:             **ad1262\_reset** contains the returned integer error code:

0 = no error

Example:            The following C program segment resets all RTI-1262 cards in a system.

```
errcode=ad1262_reset();
```



# PRO-LOG

## 75XX,76XX,and 79XX

### Input/Output Cards

#### C library

The Pro-Log input/output cards are general-purpose, low-cost analog I/O cards. The pl7000 library routines are general-purpose I/O routines that can be used with the following Pro-Log cards: 7501, 7502, 7503, 7504, 7506, 7601A, 7602A, 7603A, 7604A, 7605, 7614, 7904, and 7914. Following the description of the STD-LIB routines is a description of each of the Pro-Log I/O cards that can be used with the library software. Shown below is a chart that shows which pl7000 routines can be used with each I/O card.

|                  | 7501 | 7502 | 7503 | 7504 | 7506 | 7601A | 7602A | 7603A | 7604A | 7605 | 7614 | 7904 | 7914 |
|------------------|------|------|------|------|------|-------|-------|-------|-------|------|------|------|------|
| pl7000_init      | *    | *    | *    | *    | *    | *     | *     | *     | *     | *    | *    | *    | *    |
| pl7000_rdbit     |      |      | *    |      | *    | *     |       | *     | *     | *    | *    | *    | *    |
| pl7000_rdport    |      |      | o    |      | o    | *     | *     | *     | *     | *    | *    | *    | *    |
| pl7000_resetport | *    | *    |      | *    |      | *     | *     | *     | *     | *    | *    | *    | *    |
| pl7000_wrtbit    | *    | *    |      | *    |      | *     | *     | *     | *     | *    | *    | *    | *    |
| pl7000_wrtbitrd  |      |      |      |      |      |       |       | *     | *     | *    | *    | *    | *    |
| pl7000_wrtport   | o    | o    |      | o    |      | *     | *     | *     | *     | *    | *    | *    | *    |
| pl7000_wrtportrd |      |      |      |      |      |       |       | *     | *     | *    | *    | *    | *    |

\* = Routine can be used for card

o = Routine can be used for card, but other routine is typically preferred

The STD-LIB 7000 software modules are as follows:

|                          |                                                                                                          |
|--------------------------|----------------------------------------------------------------------------------------------------------|
| <b>pl7000_init:</b>      | Initializes the library software with the user-selected hardware configuration of the card.              |
| <b>pl7000_rdbit:</b>     | Reads a bit from the selected port.                                                                      |
| <b>pl7000_rdport:</b>    | Reads an 8-bit value from the selected port.                                                             |
| <b>pl7000_resetport:</b> | Software resets a group of contiguously mapped output ports by writing a 0 to all ports.                 |
| <b>pl7000_wrtbit:</b>    | Writes a 1 or a 0 to the selected bit of a port.                                                         |
| <b>pl7000_wrtbitrd:</b>  | Writes a 1 or a 0 to the specified bit and port, then reads back the bit value to verify a proper write. |
| <b>pl7000_wrtport:</b>   | Writes a specified 8-bit pattern to a port.                                                              |
| <b>pl7000_wrtportrd:</b> | Writes a specified 8-bit pattern to a port, then reads back an 8-bit value to verify a proper write.     |

Structure:           extern int pl7000\_\_init();  
                      int baseno;  
                      int baseadr;  
                      int nocards;  
                      int noports;  
                      pl7000\_\_init(baseno,baseadr,nocards,noports);

Description:           This routine initializes the pl7000 library for port I/O operations with the user-selected hardware configuration, if different from the default condition. The default is for one card (card 0) in a system containing two contiguously mapped ports with the lowest port address at hexadecimal 040. When using more than one card, all cards with the same configuration are mapped contiguously and are initialized with the same routine call.

Input  
Parameters:           **baseno** is an integer value specifying the base number of the group of contiguous cards configured by this command (0 -21).  
  
                      **baseadr** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the contiguously mapped pl7000 cards.  
  
                      **nocards** is an integer value specifying the number of contiguously mapped cards specified by this command (1 -22).  
  
                      **noports** is an integer value specifying the number of ports on each card (1 - 8).

## pl7000\_\_init

Returned  
Value:

pl7000\_\_init contains the returned integer  
error code:

- 0 = no error
- 1 = board number out of range
- 2 = invalid card count (number of cards > 22,  
or the number of cards causes the base  
address of the last card to be less than  
that of the first card (wraparound))
- 3 = invalid number of ports

Example:

The following C program segment could be used to initialize the library for a group of four 7501 cards after a group of three differently configured cards have already been initialized. This group of cards is mapped contiguously beginning at 8-bit I/O address 0x20. The cards are configured for 2 ports per card.

```
errcode=pl7000__init(3,0x120,4,2);
```

75XX/76XX/79XX

C library

---

|                   |                                                                                                                                                                                                                                                                                              |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7000_rdbit(); int cardno; int portno; int bitno; pl7000_rdbit(cardno,portno,bitno);</pre>                                                                                                                                                                                  |
| Description:      | This routine reads a bit from a port and returns 0, bit cleared, or 1 bit set.                                                                                                                                                                                                               |
| Input Parameters: | <p><b>cardno</b> is an integer value specifying the card to read (0 - 21).</p> <p><b>portno</b> is an integer value specifying the port to be read (0 - 7).</p> <p><b>bitno</b> is an integer value specifying the bit to be read (0 - 7).</p>                                               |
| Returned Value:   | <p><b>pl7000_rdbit</b> contains the returned bit value or integer error code:</p> <ul style="list-style-type: none"><li>1 = bit is set</li><li>0 = bit is cleared</li><li>-1 = card number out of range</li><li>-2 = port number out of range</li><li>-3 = bit number out of range</li></ul> |
| Example:          | <p>The following C program segment could be written to read bit four of the second port and the second card in the system.</p> <pre>errcode=pl7000_rdbit(1,1,4);</pre>                                                                                                                       |

|                      |                                                                                                                                                                                                                    |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:           | <pre>extern int pl7000_rdport();<br/>int cardno;<br/>int portno;<br/>pl7000_rdport(cardno,portno);</pre>                                                                                                           |
| Description:         | This routine reads an 8-bit value from a port.                                                                                                                                                                     |
| Input<br>Parameters: | <p><b>cardno</b> is an integer value specifying the card to be read (0 - 21).</p> <p><b>portno</b> is an integer value specifying the port to be read (0 - 7).</p>                                                 |
| Returned<br>Value:   | <p><b>pl7000_rdport</b> contains the port data (0-255) or the returned integer error code:</p> <ul style="list-style-type: none"><li>-1 = card number out of range</li><li>-2 = port number out of range</li></ul> |
| Example:             | <p>The following C program segment could be written to read port 4 of the middle card in a three-card system.</p> <pre>errcode=pl7000_rdport(1,4);</pre>                                                           |

75XX/76XX/79XX

C library

Structure:           extern int pl7000\_resetport();  
                      int baseno  
                      pl7000\_resetport(baseno);

Description:       This routine software resets a group of  
                     contiguously mapped output ports. This group  
                     of cards was defined by pl7000\_init. All ports  
                     in this group of contiguous cards should be  
                     output ports or input/output ports. A zero is  
                     written to all the ports.

Input  
Parameters:       **baseno** is the base number of the group of  
                     contiguous cards reset by this command  
                     (0 - 21).

Returned  
Value:           **pl7000\_resetport** contains the returned integer  
                     error code:

                  0 = no error  
                  -1 = base number out of range

Example:           The following C program segment resets a  
                     group of contiguously mapped cards beginning  
                     at card 0.

```
errcode=pl7000_resetport(0);
```

|                   |                                                                                                                                                                                                                                                                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7000_wrtbit(); int cardno; int portno; int bit; int bitno; pl7000_wrtbit(cardno,portno,bit,bitno);</pre>                                                                                                                                                                                                                       |
| Description:      | This routine writes a 1 or a 0 to a specified bit of a port.                                                                                                                                                                                                                                                                                     |
| Input Parameters: | <p><b>cardno</b> is an integer value specifying the card to be written to (0 - 21).</p> <p><b>portno</b> is an integer value specifying the port to be written to (0 - 7).</p> <p><b>bit</b> is the data to be written to the bit (0 or 1).</p> <p><b>bitno</b> is an integer value specifying the bit of the port being written to (0 - 7).</p> |
| Returned Value:   | <p><b>pl7000_wrtbit</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid card selected</li><li>-2 = port number out of range</li><li>-3 = bit number out of range</li><li>-4 = invalid write data</li></ul>                                                                 |
| Example:          | <p>The following C program segment sets bit 0 of the second port on card 2 of the system.</p> <pre>errcode=pl7000_wrtbit(2,1,1,0);</pre>                                                                                                                                                                                                         |

|                   |                                                                                                                                                                                                                                                                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7000_wrtbitrd(); int cardno; int portno; int bit; int bitno; pl7000_wrtbitrd(cardno,portno,bit,bitno);</pre>                                                                                                                                                                                                                   |
| Description:      | This routine writes a 1 or a 0 to a specified bit of a port, then reads back the bit value to verify a proper write. This routine can only be used by cards with the readback capability.                                                                                                                                                        |
| Input Parameters: | <p><b>cardno</b> is an integer value specifying the card to be written to (0 - 21).</p> <p><b>portno</b> is an integer value specifying the port to be written to (0 - 7).</p> <p><b>bit</b> is the data to be written to the bit (0 or 1).</p> <p><b>bitno</b> is an integer value specifying the bit of the port being written to (0 - 7).</p> |
| Returned Value:   | <p><b>pl7000_wrtbitrd</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid card selected</li><li>-2 = port number out of range</li><li>-3 = bit number out of range</li><li>-4 = invalid write data</li><li>-5 = read back error</li></ul>                                  |

## pl7000\_wrtbitrd

Example:

The following C program segment sets bit 0 of the second port on card 2 of the system.

```
errcode=pl7000_wrtbitrd(2,1,1,0);
```

---

|                   |                                                                                                                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7000_wrtport(); int cardno; int portno; int portdata; pl7000_wrtport(cardno,portno,portdata);</pre>                                                                                                                                           |
| Description:      | This routine writes an 8-bit pattern to a specified port.                                                                                                                                                                                                       |
| Input Parameters: | <p><b>cardno</b> is an integer value specifying the card to be written to (0 - 21).</p> <p><b>portno</b> is an integer value specifying the port to be written to (0 - 7).</p> <p><b>portdata</b> is the data to be written to the port (0 hex - 0xFF hex).</p> |
| Returned Value:   | <p><b>pl7000_wrtport</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid card selected</li><li>-2 = port number out of range</li></ul>                                                    |
| Example:          | <p>The following C program segment writes 4B hex to port 0 of the second card of the system.</p> <pre>errcode=pl7000_wrtport(1,0,0x4B);</pre>                                                                                                                   |

---

|                   |                                                                                                                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7000_wrtportrd(); int cardno; int portno; int portdata; pl7000_wrtportrd(cardno,portno,portdata);</pre>                                                                                                                                       |
| Description:      | This routine writes an 8-bit pattern to a specified port, then reads back an 8-bit value to verify a proper write. This routine can only be used by cards with the readback capability.                                                                         |
| Input Parameters: | <p><b>cardno</b> is an integer value specifying the card to be written to (0 - 21).</p> <p><b>portno</b> is an integer value specifying the port to be written to (0 - 7).</p> <p><b>portdata</b> is the data to be written to the port (0 hex - 0xFF hex).</p> |
| Returned Value:   | <p><b>pl7000_wrtportrd</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid card selected</li><li>-2 = port number out of range</li><li>-3 = readback error</li></ul>                      |
| Example:          | <p>The following C program segment writes 4B hex to port 0 of the second card of the system.</p> <pre>errcode=pl7000_wrtportrd(1,0,0x4B);</pre>                                                                                                                 |

The following Pro-Log I/O cards may be used with the STD-LIB software:

## 7501 DC DRIVER CARD

The 7501 converts TTL level signals supplied by the STD BUS to latched, negative true, open-collector DC drive signals. The 7501 provides sixteen independent DC output circuits. Each open collector output circuit is capable of sinking up to 225 mA of current in the on state and can withstand a 50V output level in the off state. The two output ports are cleared by SYSRESET\* and are consecutively mapped at addresses 40 and 41 as shipped.

The 7501 uses the general-purpose I/O routines of STD-LIB as follows:

|                    |                                                                                 |
|--------------------|---------------------------------------------------------------------------------|
| pl7000__init:      | Initializes the library software with the user-selected hardware configuration. |
| pl7000__resetport: | Software resets the output ports.                                               |
| pl7000__wrtbit:    | Writes a 1 or a 0 to a specified bit of a port.                                 |

## 7502 SPST RELAY OUTPUT CARD

The 7502 consists of eight independent SPST dry reed relays controlled by a fully decoded, latched output port. The output port is cleared by SYSRESET\* and is mapped at address 40 as shipped.

The 7502 uses the general-purpose I/O routines of STD-LIB as follows:

|                    |                                                                                 |
|--------------------|---------------------------------------------------------------------------------|
| pl7000__init:      | Initializes the library software with the user-selected hardware configuration. |
| pl7000__resetport: | Software resets the output ports.                                               |
| pl7000__wrtbit:    | Writes a 1 or a 0 to a specified bit of the port.                               |

## 7503 OPTOISOLATED INPUT CARD

The 7503 converts an AC or DC input voltage into a TTL-level signal that is read by an on-card STD BUS input port. The 7503 provides eight independent AC/DC inputs. An optical coupling circuit isolates the circuits from each other and from the STD BUS system. The 7503 offers a choice of two input ranges. Range selection is independent for each input and is selected through the installation of wire jumpers. The low range responds to input voltages between 4.5 VRMS and 22 VRMS for 5V, 6V, and 12V applications. The high range responds to input voltages of between 20 VRMS and 80 VRMS and is useful for systems using 24V, 28V, and 48V. The input port is mapped at address 40 as shipped.

The 7503 uses the general-purpose I/O routines of STD-LIB as follows:

|                            |                                                                                 |
|----------------------------|---------------------------------------------------------------------------------|
| <code>pl7000_init:</code>  | Initializes the library software with the user-selected hardware configuration. |
| <code>pl7000_rdbit:</code> | Reads a bit from the port.                                                      |

## 7504 TRIAC OUTPUT CARD

The 7504 consists of eight independent solid-state AC relays (triacs) controlled by a fully decoded, latched 8-bit output port. The 7504 allows the processor direct control of eight switched AC power circuits (40 - 280 VRMS and 47 - 63 Hz). The output port is cleared by `SYSRESET*` and is mapped at address 40 as shipped.

The 7504 uses the general-purpose I/O routines of STD-LIB as follows:

|                                |                                                                                 |
|--------------------------------|---------------------------------------------------------------------------------|
| <code>pl7000_init:</code>      | Initializes the library software with the user-selected hardware configuration. |
| <code>pl7000_resetport:</code> | Software resets the output ports.                                               |
| <code>pl7000_wrtbit:</code>    | Writes a 1 or a 0 to a specified bit of the port.                               |

## 7506 OPTOISOLATED INPUT CARD

The 7506 converts an AC or DC input voltage into a TTL-level signal that is read by an on-card STD BUS input port. The 7506 provides eight independent AC/DC inputs. An optical coupling circuit isolates the circuits from each other and from the STD BUS system. The 7506 offers a choice of two input ranges. Range selection is independent for each input and is selected through the installation of wire jumpers. The low range responds to input voltages between 70 VRMS and 150 VRMS for 120V applications. The high range responds to input voltages of between 140 VRMS and 280 VRMS and is useful for systems using 240V. The input port is mapped at address 40 as shipped.

The 7506 uses the general-purpose I/O routines of STD-LIB as follows:

|                            |                                                                                 |
|----------------------------|---------------------------------------------------------------------------------|
| <code>pl7000_init:</code>  | Initializes the library software with the user-selected hardware configuration. |
| <code>pl7000_rdbit:</code> | Reads a bit from the port.                                                      |

## 7601A TTL INPUT/OUTPUT PORT CARD

The 7601A provides four 8-bit gated input ports (32 input lines) and four 8-bit latched output ports (32 output lines). The four consecutive independent input ports and output ports are mapped at address 00 - 03 as shipped. The output ports are cleared by SYSRESET\*.

The 7601A uses the general-purpose I/O routines of STD-LIB as follows:

|                    |                                                                                 |
|--------------------|---------------------------------------------------------------------------------|
| pl7000__init:      | Initializes the library software with the user-selected hardware configuration. |
| pl7000__rdbit:     | Reads a bit from a port.                                                        |
| pl7000__rdport:    | Reads an 8-bit value from a port.                                               |
| pl7000__resetport: | Software resets the output ports.                                               |
| pl7000__wrtbit:    | Writes a 1 or a 0 to a specified bit of a port.                                 |
| pl7000__wrtport:   | Writes an 8-bit pattern to a port.                                              |

## 7602A TTL OUTPUT PORT CARD

The 7602A provides eight 8-bit latched output ports (64 lines). The consecutively mapped ports are cleared by SYSRESET\* and are mapped at address 00 -07 as shipped.

The 7602A uses the general-purpose I/O routines of STD-LIB as follows:

|                   |                                                                                 |
|-------------------|---------------------------------------------------------------------------------|
| pl7000_init:      | Initializes the library software with the user-selected hardware configuration. |
| pl7000_resetport: | Software resets the output ports.                                               |
| pl7000_wrtbit:    | Writes a 1 or a 0 to a specified bit of a port.                                 |
| pl7000_wrtport:   | Writes an 8-bit pattern to a port.                                              |

## 7603A TTL INPUT PORT CARD

The 7603A provides eight 8-bit gated input ports (64 input lines). The consecutive ports are mapped at address 00 - 07 as shipped.

The 7603A uses the general-purpose I/O routines of STD-LIB as follows:

- pl7000\_init:      Initializes the library software with the user-selected hardware configuration.
- pl7000\_rdbit:    Reads a bit from a port.
- pl7000\_rdport:   Reads an 8-bit value from a port.

## 7604A TTL INPUT/OUTPUT CARD

The 7604A provides up to eight ports of TTL I/O. The eight ports can be individually configured as input, output, or output with readback. The consecutive ports are mapped at address 00 - 07 as shipped. The output ports are cleared by SYSRESET\*.

The 7604A uses the general-purpose I/O routines of STD-LIB as follows:

|                   |                                                                                                         |
|-------------------|---------------------------------------------------------------------------------------------------------|
| pl7000_init:      | Initializes the library software with the user-selected hardware configuration.                         |
| pl7000_rdbit:     | Reads a bit from a port.                                                                                |
| pl7000_rdport:    | Reads an 8-bit value from a port.                                                                       |
| pl7000_resetport: | Software resets the output ports.                                                                       |
| pl7000_wrtbit:    | Writes a 1 or a 0 to a specified bit of a port.                                                         |
| pl7000_wrtbitrd:  | Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write. |
| pl7000_wrtport:   | Writes an 8-bit pattern to a port.                                                                      |
| pl7000_wrtportrd: | Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.              |

## 7605 PROGRAMMABLE TTL I/O CARD

The 7605 is a general-purpose TTL I/O card that provides four 8-bit I/O ports. Since each of the 32 I/O lines can be individually programmed as input, output, or output with readback, the 7605 is particularly useful when an external device requires more or less than eight lines or a mixture of inputs and outputs. The four consecutive ports are mapped at address 00 - 03 as shipped. The output ports are cleared by SYSRESET\*.

The 7605 uses the general-purpose I/O routines of STD-LIB as follows:

|                    |                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------|
| pl7000__init:      | Initializes the library software with the user-selected hardware configuration.                        |
| pl7000__rdbit:     | Reads a bit from a port.                                                                               |
| pl7000__rdport:    | Reads an 8-bit value from a port.                                                                      |
| pl7000__resetport: | Software resets the output ports.                                                                      |
| pl7000__wrtbit:    | Writes a 1 or a 0 to a specified bit of a port.                                                        |
| pl7000__wrtbitrd:  | Writes a 1 or a 0 to a specified bit of a port and then reads back the value to verify a proper write. |
| pl7000__wrtport:   | Writes an 8-bit pattern to a port.                                                                     |
| pl7000__wrtportrd: | Writes an 8-bit pattern to a port and then reads back the value to verify a proper write.              |

## 7614 64-LINE TTL I/O CARD

The 7614 provides eight 8-bit I/O ports, 64 I/O lines total. Each port can be configured as input, output, or output with readback. The 7614 operates in systems using an 8- or 10-bit I/O address. The I/O ports can be configured as four or eight ports, and are mapped at address 00 - 07 as shipped. The output ports are cleared by SYSRESET\*.

The 7614 uses the general-purpose I/O routines of STD-LIB as follows:

|                                |                                                                                                         |
|--------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>pl7000_init:</code>      | Initializes the library software with the user-selected hardware configuration.                         |
| <code>pl7000_rdbit:</code>     | Reads a bit from a port.                                                                                |
| <code>pl7000_rdport:</code>    | Reads an 8-bit value from a port.                                                                       |
| <code>pl7000_resetport:</code> | Software resets the output ports.                                                                       |
| <code>pl7000_wrtbit:</code>    | Writes a 1 or a 0 to a specified bit of a port.                                                         |
| <code>pl7000_wrtbitrd:</code>  | Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write. |
| <code>pl7000_wrtport:</code>   | Writes an 8-bit pattern to a port.                                                                      |
| <code>pl7000_wrtportrd:</code> | Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.              |

## 7904 DECODED I/O UTILITY CARD

The 7904 is a printed circuit card designed for prototyping I/O circuitry. It provides complete STD BUS buffering and decoding for I/O cards with input and output port strobe logic and access to memory page and CPU/memory control lines. The card is configured for eight consecutive I/O ports mapped at address 00 - 07 as shipped.

The 7904 can use the appropriate general-purpose I/O routines of STD-LIB as follows:

|                   |                                                                                                         |
|-------------------|---------------------------------------------------------------------------------------------------------|
| pl7000_init:      | Initializes the library software with the user-selected hardware configuration.                         |
| pl7000_rdbit:     | Reads a bit from a port.                                                                                |
| pl7000_rdport:    | Reads an 8-bit value from a port.                                                                       |
| pl7000_resetport: | Software resets the output ports.                                                                       |
| pl7000_wrtbit:    | Writes a 1 or a 0 to a specified bit of a port.                                                         |
| pl7000_wrtbitrd:  | Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write. |
| pl7000_wrtport:   | Writes an 8-bit pattern to a port.                                                                      |
| pl7000_wrtportrd: | Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.              |

## 7914 DECODED I/O UTILITY CARD

The 7914 is a printed circuit card designed for prototyping I/O circuitry. The card provides STD BUS buffering and I/O port select decoding. The card is configured for eight consecutive I/O ports mapped at address 0000 - 0007 as shipped.

The 7914 can use the appropriate general-purpose I/O routines of STD-LIB as follows:

|                                 |                                                                                                         |
|---------------------------------|---------------------------------------------------------------------------------------------------------|
| <code>pl7000__init:</code>      | Initializes the library software with the user-selected hardware configuration.                         |
| <code>pl7000__rdbit:</code>     | Reads a bit from a port.                                                                                |
| <code>pl7000__rdport:</code>    | Reads an 8-bit value from a port.                                                                       |
| <code>pl7000__resetport:</code> | Software resets the output ports.                                                                       |
| <code>pl7000__wrtbit:</code>    | Writes a 1 or a 0 to a specified bit of a port.                                                         |
| <code>pl7000__wrtbitrd:</code>  | Writes a 1 or a 0 to a specified bit of a port, and then reads back the value to verify a proper write. |
| <code>pl7000__wrtport:</code>   | Writes an 8-bit pattern to a port.                                                                      |
| <code>pl7000__wrtportrd:</code> | Writes an 8-bit pattern to a port, and then reads back the value to verify a proper write.              |



The 7303 Keyboard Display Card is a general-purpose control panel card with data input and display capability. The card includes an 8-character alphanumeric display, keyboard with 24 program-definable keys plus system reset, an 8-bit binary LED display, and two rocker switches. The alphanumeric displays do not display lower case characters. The display routines convert characters to upper case.

The STD-LIB 7303 software modules are as follows:

- |                         |                                                                                                              |
|-------------------------|--------------------------------------------------------------------------------------------------------------|
| <b>pl7303_chrshft:</b>  | Shifts the displayed characters to the left and brings in a new character at the right.                      |
| <b>pl7303_chrw:</b>     | Writes an ASCII character to a specified display position.                                                   |
| <b>pl7303_cursor:</b>   | Sets or removes one or more cursors from the display.                                                        |
| <b>pl7303_getkey:</b>   | Waits for and then reads a keystroke. The routine waits for a new keystroke and then waits for key debounce. |
| <b>pl7303_getrockr:</b> | Reads a valid rocker switch state. The routine waits for any switch bouncing to finish.                      |

|                         |                                                                                                                                                                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pl7303_init:</b>     | Initializes the library software with the user-selected hardware configuration of the card. This routine also performs self-test functions: a write to the displays and LEDs for visual check, and a keyboard check for a "stuck key." |
| <b>pl7303_keyfetch:</b> | To be called 15 ms after the pl7303_keystat routine. It returns a key if bouncing has ended and a new keystroke has been detected.                                                                                                     |
| <b>pl7303_keystat:</b>  | Checks to see if a new key is currently closed. The routine does not indicate whether the depressed key has stopped bouncing.                                                                                                          |
| <b>pl7303_message:</b>  | Shifts a message into the display from the right and continues shifting the message across until the display is blank.                                                                                                                 |
| <b>pl7303_rckrstat:</b> | Returns the rocker switch status: open, closed, or changed.                                                                                                                                                                            |
| <b>pl7303_reset:</b>    | Blanks all displays.                                                                                                                                                                                                                   |
| <b>pl7303_setleds:</b>  | Sets the state of the LEDs.                                                                                                                                                                                                            |
| <b>pl7303_test:</b>     | Performs a self-test of the card, including a write to the displays and LEDs for visual check, and a keyboard check for a "stuck key."                                                                                                 |
| <b>pl7303_wait:</b>     | Waits 15 ms for debouncing a key in the keyboard or a rocker switch.                                                                                                                                                                   |
| <b>pl7303_word:</b>     | Displays an 8-character (or shorter) word from left to right and blank fills any remaining portion of the display.                                                                                                                     |

---

Structure:           extern int pl7303\_chrshft();  
                      char chr;  
                      pl7303\_chrshft(chr);

Description:        This routine shifts the displayed characters to the left and brings in a new character at the right.

Input  
Parameters:         **chr** is the ASCII character to be shifted into the display at the right.

Returned  
Value:             **pl7303\_chrshft** contains the returned integer error code:

                    0 = no error  
                    -1 = no 7303 installed  
                    -2 = invalid character selected

Example:            The following C program segment could be written to shift the display left and bring in an "A" to the rightmost display.

                     errcode=pl7303\_chrshft(A);

---

|                   |                                                                                                                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7303_chrw();<br/>char chr;<br/>int dispno;<br/>pl7303_chrw(chr,dispno);</pre>                                                                                                                                               |
| Description:      | This routine writes an ASCII character to the specified display.                                                                                                                                                                              |
| Input Parameters: | <p><b>chr</b> is an ASCII character to be written to a display.</p> <p><b>dispno</b> is an integer value specifying the display to be written (0 - 7). The rightmost display is defined as display 0.</p>                                     |
| Returned Value:   | <p><b>pl7303_chrw</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = no 7303 installed</li><li>-2 = invalid display selected</li><li>-3 = invalid character selected</li></ul> |
| Example:          | <p>The following C program segment could be written to write an "A" to the third display from the left.</p> <pre>errcode=pl7303_chrw(A,5);</pre>                                                                                              |

---

|                   |                                                                                                                                                                                                                                                            |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7303_cursor();<br/>int display;<br/>int onoff;<br/>pl7303_cursor(display,onoff);</pre>                                                                                                                                                   |
| Description:      | <p>This routine writes a cursor (all segments lit) to the specified displays. With a valid ASCII character loaded to a display position, the cursor can also be displayed in that position. When the cursor is removed, the ASCII character reappears.</p> |
| Input Parameters: | <p><b>display</b> is a binary integer number indicating which displays are to be written to.</p> <p><b>onoff</b> is a binary integer number specifying whether the associated cursor position is to be turned on or off (1 = on, 0 = off).</p>             |
| Returned Value:   | <p><b>pl7303_cursor</b> contains the returned integer error code:</p> <p>0 = no error<br/>-1 = no 7303 installed</p>                                                                                                                                       |
| Example:          | <p>The following C program segment could be written to write a cursor to the third display from the left and to turn off the rightmost cursor.</p> <pre>errcode=pl7303_cursor(0x21,0x20);</pre>                                                            |

Structure:           extern int pl7303\_getkey();  
                      pl7303\_getkey();

Description:        This routine waits for a keystroke and then  
                      waits for key debounce. The routine then  
                      returns with the key that was pressed.

Input  
Parameters:         None

Returned  
Value:             pl7303\_getkey contains the key that  
                      was detected or an error code:

                      0x0-0x17 = key detected  
                      -1 = no 7303 installed

Example:            The following C program segment could be  
                      written to read a key from the keypad.

                      key=pl7303\_getkey();

---

Structure:           extern int pl7303\_getrockr();  
                      pl7303\_getrockr();

Description:        This routine reads a valid rocker switch state.  
                      The routine waits for any switch bouncing to  
                      end.

Input  
Parameters:         None

Returned  
Value:             **pl7303\_getrockr** contains the  
                      returned rocker switch status:

0 = both switches open  
1 = SW1 closed, SW2 open  
2 = SW2 closed, SW1 open  
3 = both switches closed  
-1 = no 7303 installed

Example:            The following C program segment could be  
                      written to get the valid status of rocker switch 2.

```
stat=pl7303_getrockr();
```

---

Structure:           extern int pl7303\_init();  
                      int baseadr;  
                      pl7303\_init(baseadr);

Description:       This routine initializes the 7303 library routines with the configuration of the card if different from the as-shipped condition (baseadr = 0xD0). The storage for the previous condition of the rocker switches is initialized to open. There may be only one 7303 card in a system. This routine also lights all LEDs and all the segments on the displays for a one-second visual check of proper operation. A "stuck key" test is performed, in which the keyboard is checked to see that no key is depressed. All LEDs and displays are then blanked.

Input  
Parameters:       **baseadr** is a hexadecimal integer value specifying the base address of the card. This is the lowest port address of the 7303 card. The library routines use a 10-bit address, even though the card can only be set to an 8-bit address. Set the two most significant bits of the 10-bit address to 01 (for example, 7303 address D0 is entered as 1D0).

Returned  
Value:

**pl7303\_\_init** contains the returned  
integer error code:

0 = no error  
-1 = no 7303 installed  
other = row of stuck key (binary  
coded)  
bit 0=row1:keys 0-3  
bit 1=row2:keys 4-7  
bit 2=row3:keys 8-B  
bit 3=row4:keys C-F  
bit 4=row5:keys 10-13  
bit 5=row6:keys 14-17

Example:

The following C program segment could be  
written to initialize a 7303 card mapped at 0xC0.

```
baseadr = 0xC0;
errcode=pl7303__init(baseadr);
```

## pl7303\_keyfetch

7303

C library

---

|                   |                                                                                                                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <code>extern int pl7303_keyfetch();<br/>pl7303_keyfetch();</code>                                                                                                                                                          |
| Description:      | This routine reads the status of the keypad. It is called 15 ms after the pl7303_keystat routine.                                                                                                                          |
| Input Parameters: | None                                                                                                                                                                                                                       |
| Returned Value:   | <p><b>pl7303_keyfetch</b> contains the returned keypad status. 0x0 - 0x17 for the key number if a valid key depression was detected.</p> <p>-1 = no 7303 installed<br/>-2 = no valid key detected</p>                      |
| Example:          | <p>The following C program segment could be written to read the key status (typically written after pl7303_keystat and pl7303_wait routines or their equivalents have been called).</p> <pre>stat=pl7303_keyfetch();</pre> |

7303

C library

Structure:           extern int pl7303\_\_keystat();  
                          pl7303\_\_keystat();

Description:        This routine checks to see if a new key on the keypad is currently closed. The routine does not indicate whether the depressed key has stopped bouncing.

Input  
Parameters:         None

Returned  
Value:             **pl7303\_\_keystat** contains the returned keypad status:

0 = no new key closed  
1 = new key closed  
-1 = no 7303 installed

Example:            The following C program segment could be written to get the status of the keypad.

```
stat=pl7303__keystat();
```

---

|                   |                                                                                                                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7303_message ();<br/>pl7303_message(msg);</pre>                                                                                                                                                                     |
| Description:      | This routine writes an ASCII message to the displays from the right. The message is shifted through the displays until all the displays are left blank. Any non-ASCII characters found in the string are replaced with a space.       |
| Input Parameters: | <b>msg</b> is a pointer to the message to be displayed. The message must have between 1 and 65535 characters.                                                                                                                         |
| Returned Value:   | <b>pl7303_message</b> contains the returned integer error code:<br><br>0 = no error<br>-1 = no 7303 installed                                                                                                                         |
| Example:          | <p>The following C program segment could be written to scroll a message (stored at location <b>msg</b>) through the displays.</p> <pre>char msg [100]="PRINT DEMONSTRATION";<br/>.<br/>.<br/>.<br/>errcode=pl7303_message(msg);</pre> |

7303

C library

Structure:           extern int pl7303\_rckrstat();  
                      pl7303\_rckrstat();

Description:       This routine reads the current status of a rocker switch (open, closed, or changed). The routine does not indicate whether a rocker that has changed status since the last call has stopped bouncing.

Input  
Parameters:       None

Returned  
Value:            pl7303\_rckrstat contains the returned rocker switch status:

- 1 = SW1 status change
- 2 = SW2 status change
- 3 = SW1 and SW2 status change
- 4 = no status change, both switches open
- 5 = no status change, SW1 closed, SW2 open
- 6 = no status change, SW2 closed, SW1 open
- 7 = no status change, both switches closed
- 1 = no 7303 installed

Example:           The following C program segment could be written to get the status of the rocker switches.

```
stat=pl7303_rckrstat();
```

Structure:           extern int pl7303\_reset();  
                      pl7303\_reset();

Description:        This routine software resets the 7303. All alphanumeric and LED displays are blanked.

Input  
Parameters:         None

Returned  
Value:             pl7303\_reset contains the returned integer error code:

                    0 = no error  
                    -1 = no 7303 installed

Example:            The following C program segment resets the 7303.

```
errcode=pl7303_reset();
```

---

|                   |                                                                                                                                                                                                                        |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7303_setleds();<br/>int leds;<br/>int onoff;<br/>pl7303_setleds(leds,onoff);</pre>                                                                                                                   |
| Description:      | This routine sets the state of the LED displays. Note that many of the other 7303 routines can affect the state of the LEDs. The LEDs are restored to their previous condition after any of these routines are called. |
| Input Parameters: | <p><b>leds</b> is a binary integer number indicating which LEDs are to be changed.</p> <p><b>onoff</b> is a binary integer number specifying the values for the LEDs that are being written (1 = on, 0 = off).</p>     |
| Returned Value:   | <p><b>pl7303_setleds</b> contains the returned integer error code:</p> <p>0 = no error<br/>-1 = no 7303 installed</p>                                                                                                  |
| Example:          | <p>The following C program segment could be written to set the third LED from the left and clear the fifth LED from the left.</p> <pre>errcode=pl7303_setleds(0x28,0x20);</pre>                                        |

Structure:           extern int pl7303\_\_test();  
                      pl7303\_\_test();

Description:        This routine performs a self-test of the 7303 card. It lights all LEDs and all the segments on the displays for a one-second visual check of proper operation. A "stuck key" test is performed, in which the keyboard is checked to see that no key is depressed. All LEDs and displays are then blanked.

Input

Parameters:        None

Returned  
Value:            pl7303\_\_test contains the returned integer error code:

0 = no error  
-1 = no 7303 installed  
other = row of stuck key  
      (binary coded)  
      bit 0=row1:keys 0-3  
      bit 1=row2:keys 4-7  
      bit 2=row3:keys 8-B  
      bit 3=row4:keys C-F  
      bit 4=row5:keys 10-13  
      bit 5=row6:keys 14-17

Example:            The following C program segment could be written to self-test a 7303 card.

```
errcode=pl7303__test();
```

7303

C library

---

Structure:           extern int pl7303\_wait();  
                      pl7303\_wait();

Description:        This routine waits 15 ms, the time required to  
                      debounce a keyboard switch or rocker switch.

Input  
Parameters:         None

Returned  
Value:             pl7303\_wait contains the returned  
                     integer error code:

                    0 = no error  
                    -1 = no 7303 installed

Example:            The following C program segment waits 15 ms  
                     for a 7303 switch to debounce.

```
errcode=pl7303_wait();
```

---

Structure:            `extern int pl7303_word();`  
                      `pl7303_word(msg);`

Description:        This routine writes an ASCII 8-character (or shorter) word to the display from left to right. Blanks are written to any remaining portion of the display.

Input  
Parameters:         `msg` is a pointer to the word to be displayed.

Returned  
Value:              `pl7303_word` contains the returned integer error code:

                      0 = no error  
                      -1 = no 7303 installed

Example:            The following C program segment could be written to write a four-character word (stored at location `msg`) to the first four display positions, followed by four blanks.

```
char msg[100]="TEST";
.
.
.
errcode=pl7303_word(msg);
```

The 7312 and 7314A Peripheral Interface Cards provide full-duplex RS-232-C serial channels for connection to terminals, printers, modems, and other peripherals or systems. Each card features two (7312) or four (7314A) serial channels, each with selectable DCE/DTE configuration using standard RS-232-C line drivers and receivers.

Up to 32 RS-232-C channels are supported by the STD-LIB software modules. Each channel can be separately configured for baud rate, data bits, stop bits, handshake, and parity. Input and output are interrupt driven with user-defined receive and transmit buffers.

The STD-LIB 7312 and 7314A software modules are as follows:

|                         |                                                                                                  |
|-------------------------|--------------------------------------------------------------------------------------------------|
| <b>pl7314_bufflush:</b> | Removes all characters from the transmit and/or receive buffer.                                  |
| <b>pl7314_chanrst:</b>  | Initializes one RS-232-C channel.                                                                |
| <b>pl7314_init:</b>     | Initializes the library software with the user-selected hardware configuration for all channels. |
| <b>pl7314_inten:</b>    | Enables or disables receive, transmit, or break interrupts.                                      |
| <b>pl7314_rcv:</b>      | Reads one received character from the receive buffer.                                            |
| <b>pl7314_rcvchk:</b>   | Checks to see if a received character is available in the receive buffer.                        |

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>pl7314_setbuf:</code> | Allocates a transmit or receive buffer to be used in place of the default 8-character buffers allocated by <code>pl7314_init</code> . |
| <code>pl7314_test:</code>   | Tests a serial channel by writing to and reading from the control registers for the channel.                                          |
| <code>pl7314_xmt:</code>    | Transmits a character by placing it in the transmit buffer.                                                                           |
| <code>pl7314_xmtbrk:</code> | Transmits a break condition immediately.                                                                                              |
| <code>pl7314_xmtchk:</code> | Checks to see if there is room in the transmit buffer for a character.                                                                |
| <code>pl7314_xmtimd:</code> | Transmits one character immediately without using the transmit buffer.                                                                |

**NOTE:** The 7312 and 7314A as-shipped mapping is at hexadecimal address 0B0 through 0B7. When one of these cards is installed in an STD-DOS system with the as-shipped mapping, STD-DOS initializes the card as communication ports COM1 through COM4. If the card is not used for STD-DOS communication ports, the library can simply re-initialize the card. To avoid the possibility of someone attempting to use a 7312 or 7314A card for two different purposes simultaneously, the default library address for a 7312 or 7314A card is different from the address used by STD-DOS. The library default addresss is hexadecimal 160 through 167.

To change a 7312-1 or 7314A-1 card from its as-shipped mapping to the library default mapping, make the following changes to the W1 jumper header:

remove 5-6  
install 7-8  
remove 9-10  
install 13-14

Structure:               extern int pl7314\_bufflush();  
                           int chan;  
                           int xmt;  
                           int rcv;  
                           **pl7314\_bufflush(chan,xmt,rcv);**

Description:           This routine removes all characters from the transmit and/or receive buffers.

Input  
 Parameters:           **chan** is an integer value specifying the number of the channel for which the transmit and/or receiver buffer(s) is being flushed (0-31).  
  
                           **xmt** is an integer value specifying whether the transmit buffer is to be flushed (xmt=1) or not (xmt=0).  
  
                           **rcv** is an integer value specifying whether the receive buffer is to be flushed (rcv=1) or not (rcv=0).

Returned  
 Value:               **pl7314\_bufflush** contains the returned integer error code:  
  
                           0 = no error  
                           -1 = invalid channel selected

Example:               The following C program segment could be written to flush the transmit buffer for channel 4.  
  
                           **errcode=pl7314\_bufflush(4,1,0);**

Structure:           extern int pl7314\_chanrst();  
                      int chan;  
                      int baud;  
                      int parity;  
                      int bitschar;  
                      int stopbit;  
                      int handshk;  
                      **pl7314\_chanrst(chan,baud,parity,**  
                      **bitschar,stopbit,handshk);**

Description:           This routine initializes a channel and enables the transmit, receive, and break interrupts for one channel of a 7312 or 7314A card. This routine must be called once for each serial channel in the system. The pl7314\_init routine must be called before this routine can be called. The default 8-character transmit and receive buffers are allocated within the pl7314\_init module. Different size buffers can be allocated by the pl7314\_setbuf module.

Input  
Parameters:           **chan** is an integer value specifying the number of the channel that is being initialized (0 - 31).

**baud** is an integer value specifying the baud rate (19200, 9600, 4800, 2400, 1200, 600, 300, 150, 110).

**parity** is an integer value specifying parity (0 = no parity, 1 = odd parity, 2 = even parity).

**bitschar** is an integer value specifying the number of data bits per character (5, 6, 7, 8).

**stopbit** is an integer value specifying the number of stop bits (1 or 2).

**handshk** is an integer value specifying the handshake (0 = no handshake, 1 = CTS, 2 = XON/XOFF).

Returned  
Value:

**pl7314\_\_chanrst** contains the returned integer error code:

0 = no error

-1 = error, channel cannot be  
initialized as requested

Example:

The following C program segment could be written to initialize serial channel 1 of a system. The channel is initialized to 9600 baud, no parity, 8 data bits per character, one stop bit, and XON/XOFF handshake protocol.

```
errcode=pl7314__chanrst(1,9600,0,8,1,2);
```

---

Structure:

```
extern int pl7314_init();
int array[maxchan*24];
int baseadr;
int maxchan;
int vectlo;
pl7314_init(baseadr,maxchan,vectlo,
array);
```

Description:

This routine initializes the 7312 or 7314A library routines with the user-selected hardware configuration for all serial channels. All 7312 or 7314A cards must be mapped contiguously. This routine checks that all indicated cards are properly installed and mapped. This routine must be called before any other 7314 module is called.

Input Parameters:

**baseadr** is a hexadecimal integer value specifying the 10-bit base address of the cards. This is the lowest port address of the contiguously mapped 7312 and 7314A cards.

**maxchan** is an integer value specifying the number of RS-232-C channels supported by 7312/7314A routines (1 - 32).

**vectlo** is a hexadecimal integer value specifying the first interrupt vector available for receive and transmit interrupts. Starting with this interrupt, this software initializes and uses n consecutive interrupts, where  $n = \text{maxchan}/2$  (each pair of channels uses one interrupt).

**array** is a hexadecimal integer address pointing to a buffer that can be used by the 7312/7314A software. The size of the buffer in bytes must

be maxchan times 24 (24 words per channel are required).

Returned  
Value:

**pl7314\_init** contains the returned  
integer error code:

0 = no error

n = number of requested channels  
that could not be initialized  
(n is a number from 1 to 32)

Example:

The following C program segment could be written to initialize the library for ten RS-232-C channels, located on two 7314A cards and one 7312 card. Channels 0-3 are located on a 7314A mapped at 0x100-0x107, channels 4-7 on a 7314A mapped at 0x108-0x10F, and channels 8 and 9 on a 7312 mapped at 0x110-0x113. Interrupt vectors C0-C4 are available for use by this module. Memory location array contains a doubleword pointer to a 240-word RAM buffer.

```
int array[240];
errcode=pl7314_init(0x100,10,0xC0, array);
```

---

Structure:           extern int pl7314\_\_inten();  
                      int xmtint;  
                      int rcvint;  
                      int brkint;  
                      int chan;  
                      **pl7314\_\_inten(xmtint,rcvint,brkint,chan);**

Description:        This routine enables or disables receive, transmit, and break interrupts for a channel.

Input  
Parameters:        **xmtint** is an integer value specifying whether the transmitter interrupt for the channel is enabled (xmtint=1) or disabled (xmtint=0).

**rcvint** is an integer value specifying whether the receiver interrupt for the channel is enabled (rcvint=1) or disabled (rcvint=0).

**brkint** is an integer value specifying whether the break condition interrupt for the channel is enabled (brkint=1) or disabled (brkint=0). When the break interrupt is enabled, the channel responds to a break interrupt by flushing the receive buffer and placing a CNTL-C character (03H) in the receive buffer.

**chan** is an integer value specifying the number of the channel for which interrupt conditions are being defined (0-31).

Returned  
Value:

**pl7314\_inten** contains the returned  
integer error code:

0 = no error

-1 = invalid channel selected

Example:

The following C program segment could be written to enable transmitter and break condition interrupts and disable the receiver interrupt for channel 4.

```
errcode=pl7314_inten(1,0,1,4);
```

|                   |                                                                                                                                                                                                                                    |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7314_rcv();<br/>int chan;<br/>pl7314_rcv(chan);</pre>                                                                                                                                                            |
| Description:      | This routine reads a received character from the buffer if one is present, otherwise it waits until a character is received. The routine is protected from never returning (if a character isn't received) by a timeout.           |
| Input Parameters: | <b>chan</b> is an integer value specifying the number of the channel from which the character is to be received (0-31).                                                                                                            |
| Returned Value:   | <b>pl7314_rcv</b> contains the returned status of the channel:<br><br>positive number = character received<br>-1 = invalid channel selected<br>-2 = timeout error<br>-3 = parity error<br>-4 = framing error<br>-5 = overrun error |
| Example:          | The following C program segment could be written to wait for and then read a received character from channel 4.<br><br><pre>stat=pl7314_rcv(4);</pre>                                                                              |

---

|                   |                                                                                                                                                                                                                |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7314_rcvchk();<br/>int chan;<br/>pl7314_rcvchk(chan);</pre>                                                                                                                                  |
| Description:      | This routine checks a serial channel to see if a character is available in the receive buffer.                                                                                                                 |
| Input Parameters: | <b>chan</b> is an integer value specifying the number of the channel to be checked for a received character (0-31).                                                                                            |
| Returned Value:   | <b>pl7314_rcvchk</b> contains the returned status of the channel:<br><br>0 = no character available<br>positive number = number of characters available in the receive buffer<br>-1 = invalid channel selected |
| Example:          | The following C program segment could be written to check whether channel 4 has a received character available.<br><br>stat=pl7314_rcvchk(4);                                                                  |

---

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7314_setbuf(); int size; int array[size/2]; int xmtrcv; int chan; pl7314_setbuf(size,array,xmtrcv,chan);</pre>                                                                                                                                                                                                                                                                                                                                                |
| Description:      | This routine allocates a transmit or receive buffer that is a different size than the 8-character default buffer allocated by pl7314_init. An n-character receive buffer requires 2 (n + 1) bytes and an n-character transmit buffer requires (n + 1) bytes.                                                                                                                                                                                                                    |
| Input Parameters: | <p><b>size</b> is an integer value specifying the size in bytes of the buffer to be allocated (1 - 32767).</p> <p><b>array</b> is a hexadecimal integer address pointing to the buffer area.</p> <p><b>xmtrcv</b> is an integer value specifying whether the buffer is to be a transmit buffer or a receive buffer (0 = transmit, 1 = receive).</p> <p><b>chan</b> is an integer value specifying the number of the channel for which the buffer is being defined (0 - 31).</p> |
| Returned Value:   | <p><b>pl7314_setbuf</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid channel selected</li></ul>                                                                                                                                                                                                                                                                                                        |

Example:

The following C program segment could be written to allocate a 49 character receive buffer for channel 4.

```
int array[50];
errcode=pl7314__setbuf(100,array,1,4);
```

Structure:           extern int pl7314\_test();  
                      int chan;  
                      **pl7314\_test(chan);**

Description:           This routine performs a self-test on one serial channel. The routine tests the control registers of the serial chip to see if they can be written to and read from. The data registers aren't written to in this test to avoid changing any outputs of the channel.

Input Parameters:       **chan** is an integer value specifying the number of the channel to be tested (0-31).

Returned Value:       **pl7314\_test** contains the returned status of the channel:

                      0 = no error  
                      -1 = self test failed

Example:               The following C program segment could be written to test channel 4.

```
stat=pl7314_test(4);
```

7312/7314A

C library

Structure:           extern int pl7314\_xmt();  
                      int chan;  
                      char chr;  
                      pl7314\_xmt(chan,chr);

Description:           This routine transmits a character by placing it in the transmit character buffer. If there is no room in the buffer, the routine waits until there is room or until a timeout indicates a system failure.

Input  
Parameters:           **chan** is an integer value specifying the number of the channel from which the character is to be transmitted (0-31).  
  
                      **chr** is the ASCII character to be transmitted.

Returned  
Value:                **pl7314\_xmt** contains the returned status of the channel:

                      0 = no error  
                      -1 = invalid channel selected  
                      -2 = transmitter timeout error

Example:              The following C program segment could be written to transmit an ASCII "A" from channel 4.

```
stat=pl7314_xmt(4,A);
```

---

|                   |                                                                                                                                         |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7314_xmtbrk();<br/>int chan;<br/>pl7314_xmtbrk(chan);</pre>                                                           |
| Description:      | This routine transmits a break condition immediately without waiting for the characters in the transmit buffer to be sent.              |
| Input Parameters: | <b>chan</b> is an integer value specifying the number of the channel from which the break is to be transmitted (0-31).                  |
| Returned Value:   | <b>pl7314_xmtbrk</b> contains the returned status of the channel:<br><br>0 = no error<br>-1 = invalid channel selected                  |
| Example:          | The following C program segment could be written to transmit an immediate break condition from channel 4.<br><br>stat=pl7314_xmtbrk(4); |

7312/7314A

C library

---

|                   |                                                                                                                                                                                               |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7314_xmtchk();<br/>int chan;<br/>pl7314_xmtchk(chan);</pre>                                                                                                                 |
| Description:      | This routine checks to see if there is room in the transmit buffer for a character.                                                                                                           |
| Input Parameters: | <b>chan</b> is an integer value specifying the number of the channel buffer to be checked (0-31).                                                                                             |
| Returned Value:   | <b>pl7314_xmtchk</b> contains the returned status of the channel:<br><br>0 = buffer has no room<br>positive number = number of empty locations in the buffer<br>-1 = invalid channel selected |
| Example:          | <p>The following C program segment could be written to check whether channel 4 has room in the transmit buffer for another character.</p> <pre>stat=pl7314_xmtchk(4);</pre>                   |

Structure:           extern int pl7314\_xmtimd();  
                      int chan;  
                      char chr;  
                      pl7314\_xmtimd(chan,chr);

Description:           This routine transmits a character immediately  
                          without placing it in the transmit character  
                          buffer.

Input  
Parameters:           **chan** is an integer value specifying  
                          the number of the channel from which the  
                          character is to be transmitted (0-31).  
  
                      **chr** is the ASCII character to be transmitted.

Returned  
Value:                **pl7314\_xmtimd** contains the returned  
                          status of the channel:

                      0 = no error  
                      -1 = invalid channel selected

Example:              The following C program segment could be  
                          written to transmit an ASCII "A" from channel  
                          4 without using the transmit buffer.

                      stat=pl7314\_xmtimd(4,A)

**PRO-LOG**  
**7507/7508**  
**TTL Input/Output Card**

C library

The 7507 and 7508 General Purpose Interface cards allow direct connection to one (7507) or two (7508) 24-I/O module mounting racks available from Opto-22, Gordos, Crydom, etc. Module mounting racks are also available in 8-module and 16-module versions. However, the software assumes that a 7507 has only one rack connected to it, and that a 7508 has no more than two racks. This requirement is independent of rack size.

The STD-LIB 7507 and 7508 software modules are as follows:

|                        |                                                                                                                                                      |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pl7508_init:</b>    | Initializes the library software with the user-selected hardware configuration of the card.                                                          |
| <b>pl7508_rackpwr:</b> | Turns the rack power supplied to the user on or off.                                                                                                 |
| <b>pl7508_readmod:</b> | Reads an input from a module or reads the status of an output to a module.                                                                           |
| <b>pl7508_reset:</b>   | Turns the module rack power off and sets all outputs to the modules low.                                                                             |
| <b>pl7508_setmod:</b>  | Sets an output to a module to a 0 or a 1.                                                                                                            |
| <b>pl7508_test:</b>    | This is a routine that simply returns a pass indication for self-test, because the 7507 and 7508 cards can't be tested without changing the outputs. |

Structure:           extern int pl7508\_init();  
                      int baseadr;  
                      int norack;  
                      **pl7508\_init(baseadr,norack);**

Description:           This routine initializes the 7507 or 7508 library routines with the user-selected hardware configuration of the card if different from the as-shipped condition (baseadr = 0x50, norack = 2). This routine also resets the card to all module rack power switched off, all outputs low (off), and initializes storage for the previous state of the output bits to low.

Input Parameters:       **baseadr** is a hexadecimal integer value specifying the base address of the cards. This is the lowest port address of the contiguously mapped 7507 and 7508 cards. The library routines use a 10-bit address even if the card only recognizes 8 bits of address.

**norack** is an integer value specifying the number of racks controlled by contiguously mapped cards (1 - 44).

Returned Value:       **pl7508\_init** contains the returned integer error code:

0 = no error

Example:               The following C program segment could be written to initialize the library for two 7508 cards, the first card mapped at 160-167 and the second mapped at 168-16F. These cards can control four 24-position racks.

```
errcode=pl7508_init(0x160,4);
```

|                   |                                                                                                                                                                                                                               |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7508__rackpwr(); int rack; int onoff; pl7508__rackpwr(rack,onoff);</pre>                                                                                                                                    |
| Description:      | This routine is used to turn rack power on or off.                                                                                                                                                                            |
| Input Parameters: | <p><b>rack</b> is an integer value specifying the rack power to be enabled or disabled (0 - 43).</p> <p><b>onoff</b> is an integer value specifying whether rack power is to be turned on (onoff = 1) or off (onoff = 0).</p> |
| Returned Value:   | <p><b>pl7508__rackpwr</b> contains the returned integer error code:</p> <p>0 = no error<br/>-1 = invalid rack selected</p>                                                                                                    |
| Example:          | <p>The following C program segment could be written to turn on power to rack 7.</p> <pre>errcode=pl7508__rackpwr(7,1);</pre>                                                                                                  |

---

|                   |                                                                                                                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7508_readmod(); int rack; int module; pl7508_readmod(rack,module);</pre>                                                                                                      |
| Description:      | This routine reads an input module.                                                                                                                                                             |
| Input Parameters: | <p><b>rack</b> is an integer value specifying the rack containing the module to be read (0 - 43).</p> <p><b>module</b> is an integer value specifying the input module to be read (0 - 23).</p> |
| Returned Value:   | <p><b>pl7508_readmod</b> contains the results of the read:</p> <ul style="list-style-type: none"><li>0 = bit clear</li><li>1 = bit set</li><li>-1 = invalid module or rack selected</li></ul>   |
| Example:          | <p>The following C program segment could be written to read input module 22 of the middle rack in a three-rack system.</p> <pre>stat=pl7508_readmod(1,22);</pre>                                |

Structure:           extern int pl7508\_reset();  
                      pl7508\_reset();

Description:        This routine software resets all contiguously mapped 7507 or 7508 cards. All module outputs are set low (off), and all rack power is turned off.

Input  
Parameters:        None

Returned  
Value:             pl7508\_reset contains the returned integer error code:

0 = no error

Example:           The following C program segment could be written to reset all contiguously mapped 7507 or 7508 cards.

```
errcode=pl7508_reset();
```

|                   |                                                                                                                                                                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7508__setmod(); int rack; int module; int onoff; pl7508__setmod(rack,module,onoff);</pre>                                                                                                                                                                                                     |
| Description:      | This routine writes a 1 or a 0 to a module output.                                                                                                                                                                                                                                                              |
| Input Parameters: | <p><b>rack</b> is an integer value specifying the rack containing the module to be written to (0 - 43).</p> <p><b>module</b> is an integer value specifying the output module to be written to (0 - 23).</p> <p><b>onoff</b> is an integer value specifying the value to be written to the module (0 or 1).</p> |
| Returned Value:   | <p><b>pl7508__setmod</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid module or rack selected</li><li>-2 = readback error (always returned if rack power is off)</li></ul>                                                             |
| Example:          | <p>The following C program segment could be written to set output 22 of the middle rack in a three-rack system to a 1 (turn on module 22 of rack 1).</p> <pre>errcode=pl7508__setmod(1,22,1);</pre>                                                                                                             |

Structure: `extern int pl7508_test();  
pl7508_test();`

Description: This routine simply returns a pass indication for self-test, because the 7507 and 7508 cards can't be tested without changing the outputs. This routine is included because self-test routines are required for all cards in the library.

Input  
Parameters: None

Returned  
Value: **pl7508\_test** contains the returned  
integer error code:

0 = no error

Example: The following C program segment could be written to call a self-test routine for 7507 or 7508 cards.

```
pl7508_test();
```



**PRO-LOG**  
**7700**  
**Memory Diagnostic Routines**  
**C Library**

---

The Memory Diagnostic routines are designed to test all the memory cards in the STD-DOS system, this gives users a high level of confidence that their system is fully functional. The memory diagnostic routines are:

- pl7700\_checksum:** Computes and reports the checksum of a specified area of the DOS system.
- pl7700\_memtest:** Tests all system RAM and reports to the user the amount of memory (in kbytes) that was successfully tested.

PL7700

|                   |                                                                                                                                                                                                                                                           |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern long pl7700_checksum(); unsigned int segno; unsigned int nopargrf pl7700_checksum(segno,nopargrf);</pre>                                                                                                                                      |
| Description:      | This routine computes the two's complement checksum of a specified area of the DOS system. The checksum is reported as a long integer.                                                                                                                    |
| Input Parameters: | <p><b>segno</b> is an unsigned integer value specifying the start segment of the area to be checksummed (offset = 0000).</p> <p><b>nopargrf</b> is an unsigned integer value specifying the number of 16-byte paragraphs to be added to the checksum.</p> |
| Returned Value:   | <b>pl7700_checksum</b> contains the 32-bit checksum of the specified area of memory.                                                                                                                                                                      |
| Example:          | <p>The following C program segment could be written to checksum the memory from C000:0000 to C000:FFFF.</p> <pre>chksum=pl7700_checksum(0xC000,0x1000);</pre>                                                                                             |

Structure: `extern int pl7700_memtest();`  
`pl7700_memtest();`

Description: This routine tests all contiguous RAM in the STD-DOS system. The test starts at segment 0000 and continues until the top of RAM or an error is detected. For each memory location, the memory test writes, then verifies a 00 hex, then an FF hex; once tested, each memory location is restored to its original value.

Input  
Parameters: None

Returned  
Value: `pl7700_memtest` contains the amount of RAM tested successfully (in kbytes).

Example: The following C program segment could be written to test all contiguous RAM beginning at 0000:0000.

```
, membytes=pl7700_memtest();
```



The 7863 CPU card is equipped with a real-time clock device. The STD-LIB 7863 software provides the user with the ability to read or write to the clock, to wait for a specified period of time, and to set future time marks which later may be checked to see if the time has expired. All 7863 library routines except setalarm and setmark access the clock directly.

The STD-LIB 7863 software modules are as follows:

|                         |                                                                                                                                                       |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pl7863_chkmark:</b>  | Checks for expiration of a previously set time mark.                                                                                                  |
| <b>pl7863_getdate:</b>  | Gets the current date.                                                                                                                                |
| <b>pl7863_gettime:</b>  | Gets the current time.                                                                                                                                |
| <b>pl7863_getwatch:</b> | Gets the date, time, and weekday in one call.                                                                                                         |
| <b>pl7863_init:</b>     | Sets the current date and time on the clock, then resets all time marks.                                                                              |
| <b>pl7863_setalarm:</b> | Sets internal alarm marks to enable software to wait until a specific day and time.                                                                   |
| <b>pl7863_setmark:</b>  | Sets an internal future time mark for a specified time interval from the current day and time.                                                        |
| <b>pl7863_test:</b>     | Performs a self-test on the clock. The routine gets the time, waits a while, then gets the time again. It passes self-test if the time has increased. |
| <b>pl7863_wait:</b>     | Waits for a specified period of time.                                                                                                                 |
| <b>pl7863_weekday:</b>  | Gets the current day of the week.                                                                                                                     |

The clock maintains the current year in the form 0-99. For convenience, the year as specified (and returned) by all routines is in the form 1980-2079. In addition, the clock maintains the current hour in 24-hour mode (0-23). This is reported to the caller in this fashion.

This clock supports time measurement with resolution of a hundredth of a second. For the `pl7863_wait` routine, time intervals shorter than the minimum measurement will be measured using software program loops.

The `pl7863_setalarm` and `pl7863_setmark` (set interval) routines share the same internal structure. If the user attempts to set an interval mark using the same mark number as a previously set, unexpired alarm mark, the old mark will be overwritten by the new one. Marks may be reset at any time via calls to `pl7863_setmark` or `pl7863_setalarm`. It is the user's responsibility not to overwrite an existing mark before it has expired.

---

|                   |                                                                                                                                                                                                             |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7863_chkmark();<br/>int marknum;<br/>pl7863_chkmark(marknum);</pre>                                                                                                                       |
| Description:      | This routine checks for expiration of a previously set time mark. It verifies that the mark is set, then compares it with the current time. If the time is greater than the mark, the mark will be cleared. |
| Input Parameters: | <b>marknum</b> is an integer value specifying the time mark to be checked (0-9).                                                                                                                            |
| Returned Value:   | <b>pl7863_chkmark</b> contains the returned integer error code:<br><br>0 = interval has not expired<br>1 = interval has expired<br>-1 = invalid mark number<br>-2 = mark not set                            |
| Example:          | <p>The following C program segment could be written to check mark number 2 to see if it has expired.</p> <pre>errcode=pl7863_chkmark(2);</pre>                                                              |

|                      |                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:           | <pre>extern int pl7863_getdate(); int year; int month; int day; year=pl7863_getdate(&amp;month,&amp;day);</pre>                                                                       |
| Description:         | This routine gets the current date.                                                                                                                                                   |
| Input<br>Parameters: | <p><b>&amp;month</b> is a pointer to an integer where the value of month will be stored.</p> <p><b>&amp;day</b> is a pointer to an integer where the value of day will be stored.</p> |
| Returned<br>Value:   | <p><b>pl7863_getdate</b> contains the returned current year value (1980-2079).</p> <p><b>month</b> is filled (1-12).<br/><b>day</b> is filled (1-31).</p>                             |
| Example:             | <p>The following C program segment could be written to get the current date.</p> <pre>year=pl7863_getdate(&amp;month,&amp;day);</pre>                                                 |

---

|                      |                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:           | <pre>extern int pl7863_gettime(); int hours; int minutes; int hundredths; hours=pl7863_gettime(&amp;minutes, &amp;hundredths)</pre>                                                                                 |
| Description:         | This routine gets the current time.                                                                                                                                                                                 |
| Input<br>Parameters: | <p><b>&amp;minutes</b> is a pointer to an integer where the value of minutes will be stored.</p> <p><b>&amp;hundredths</b> is a pointer to an integer where the value of hundredths of a second will be stored.</p> |
| Returned<br>Value:   | <p><b>pl7863_gettime</b> contains the returned current hours (0-23).</p> <p><b>minutes</b> is filled (0-59).</p> <p><b>hundredths</b> is filled (0-5999).</p>                                                       |
| Example:             | <p>The following C program segment gets the current time.</p> <pre>hours=pl7863_gettime(&amp;minutes, &amp;hundredths);</pre>                                                                                       |

## pl7863\_getwatch

7863

C library

Structure:           extern int pl7863\_getwatch();  
                      int watcharray[8];  
                      pl7863\_getwatch(watcharray);

Description:        This routine is used to obtain the date, time, and weekday in one call. Since the data cannot easily be passed, the caller should pass the address of an eight-word array.

Input Parameters:    **watcharray** is a pointer to an eight-word array where the time data will be stored.

Returned Value:     **pl7863\_getwatch** contains the returned integer error code:

0 = no error  
-1 = pointer is illegal (all zeros)

The watcharray contains the following information in order, if no error:

year (1980-2079)  
month (1-12)  
day (1-31)  
hours (0-23)  
minutes (0-59)  
seconds (0-59)  
hundredths of seconds (0-99)  
weekday (1 = Sunday, 2 = Monday, etc.)

Example:            The following C program segment could be written to read all the time data.

```
int watcharray [8];
errcode=pl7863_getwatch (watcharray);
```

Structure:           extern int pl7863\_init();  
                      int year;  
                      int month;  
                      int day;  
                      int hours;  
                      int minutes;  
                      int seconds;  
                      int hundredths;  
                      **pl7863\_init(year,month,day,hours,**  
                      **minutes,seconds, hundredths);**

Description:           This routine sets the current date and time on the clock and clears any time marks that have been set. It also checks the clock with a call to the pl7863\_test routine. Note that this routine will change the time and date known to the STD-DOS system.

Input  
Parameters:           **year** is an integer value specifying the current year (1980-2079).  
  
                      **month** is an integer value specifying the current month (1-12)  
  
                      **day** is an integer value specifying the current day (1-31).  
  
                      **hours** is an integer value specifying the current hours (0-23).  
  
                      **minutes** is an integer value specifying the current minutes (0-59).  
  
                      **seconds** is an integer value specifying the current seconds (0-59).  
  
                      **hundredths** is an integer value specifying the current hundredths of a second (0-99).

## pl17863\_\_init

Returned  
Value:

pl17863\_\_init contains the returned integer  
error code:

- 0 = no error
- 1 = clock is not operational
- 2 = invalid date
- 3 = invalid time

Example:

The following C program segment could be written to set the current time to 07:00:00, September 1, 1986. The 7863 library software is also initialized.

```
errcode=pl17863__init(1986,9,1,7,0,0,0);
```

Structure:           extern int pl7863\_setalarm();  
                      int marknum;  
                      int weekday;  
                      int hours;  
                      int minutes;  
                      int seconds;  
                      pl7863\_setalarm(marknum,weekday,hours,  
                      minutes,seconds);

Description:           This routine sets an alarm mark for the  
                      specified time. pl7863\_chkmark can be used to  
                      determine if the alarm has occurred. Up to 10  
                      time marks may be set at a time. These ten  
                      time marks may be set by this routine or the  
                      pl7863\_setmark routine. If the specified time is  
                      earlier than the current time, the alarm will be  
                      set for the next coming week.

Input  
Parameters:           **marknum** is an integer value specifying the time  
                      mark to be set (0-9).

**weekday** is an integer value indicating the day of  
                      the week (1-7) of the alarm time.

**hours** is an integer value indicating the hour (0-  
                      23) of the alarm time.

**minutes** is an integer value indicating the  
                      minutes (0-59) of the alarm time.

**seconds** is an integer value indicating the  
                      seconds (0-59) of the alarm time.

## **pl7863\_\_setalarm**

Returned  
Value:

**pl7863\_\_setalarm** contains the returned integer  
error code:

- 0 = no error
- 1 = invalid mark number
- 2 = invalid interval

Example:

The following C program segment could be  
written to set mark number 2 for Monday, 7  
a.m.

```
errcode=pl7863__setalarm(2,2,7,0,0);
```

Structure:           extern int pl7863\_\_setmark();  
                      int marknum;  
                      int days;  
                      int hours;  
                      int minutes;  
                      int seconds;  
                      pl7863\_\_setmark(marknum, days, hours,  
                                        minutes,seconds);

Description:           This routine sets a time mark, measured from the current time, whose expiration can be checked via calls to pl7863\_\_chkmark. Up to 10 time marks may be set at a time. These ten time marks may be set by either this routine or the pl7863\_\_setalarm routine. The maximum time for a mark interval is one week.

Input Parameters:       **marknum** is an integer value specifying the time mark to be set (0-9).

**days** is an integer value indicating the number of days (0-6) in the time interval.

**hours** is an integer value specifying the number of hours (0-23) in the time interval.

**minutes** is an integer value indicating the number of minutes (0-59) in the time interval.

**seconds** is an integer value indicating the number of seconds (0-59) in the time interval.

Returned Value:       **pl7863\_\_setmark** contains the returned integer error code:

- 0 = no error
- 1 = invalid mark number
- 2 = invalid time interval

## pl7863\_\_setmark

Example:

The following C program segment could be written to set mark number 3 for one minute from now.

```
errcode=pl7863__setmark(3,0,0,1,0);
```

**Structure:**            `extern int pl7863_test();`  
                         `pl7863_test();`

**Description:**        This routine determines if the clock is operational. It will get the time, wait a while (known to be more than a hundredth of a second), and then get the time again. The clock is presumed to be operating correctly if the time has increased.

**Input**

**Parameters:**        None

**Returned Value:**     `pl7863_test` contains the returned integer error code:

0 = no error

-1 = clock is not operational

**Example:**            The following C program segment tests the clock.

```
1 errcode=pl7863_test();
```

|                   |                                                                                                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7863_wait(); int seconds; int milli; int micro; pl7863_wait(seconds,milli,micro);</pre>                                                                                                                                                                                                                       |
| Description:      | This routine can be used to wait for a specified period of time. The maximum waiting time interval is 60 seconds.                                                                                                                                                                                                               |
| Input Parameters: | <p><b>seconds</b> is an integer value indicating the number of seconds (0-59) in the wait interval.</p> <p><b>milli</b> is an integer value indicating the number of milliseconds (0-999) in the wait interval.</p> <p><b>micro</b> is an integer value indicating the number of microseconds (0-999) in the wait interval.</p> |
| Returned Value:   | <p><b>pl7863_wait</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid interval</li></ul>                                                                                                                                                                  |
| Example:          | <p>The following C program segment could be written to wait for 2.5 seconds.</p> <pre>errcode=pl7863_wait(2,500,0);</pre>                                                                                                                                                                                                       |

Structure:           extern int pl7863\_weekday();  
                      int wkday;  
                      wkday=pl7863\_weekday();

Description:           This routine gets the current day of the week.

Input  
Parameters:           None

Returned  
Value:                pl7863\_weekday contains the returned current  
                      day of the week (1-7), with 1 = Sunday, 2 =  
                      Monday, etc.

Example:              The following C program segment gets the  
                      current day of the week.

```
wkday=pl7863_weekday();
```



The 7888 CPU card is equipped with a real-time clock device. The STD-LIB 7888 software provides the user with the ability to read or write to the clock, to wait for a specified period of time, and to set future time marks which later may be checked to see if the time has expired. All 7888 library routines except setalarm and setmark access the clock directly.

The STD-LIB 7888 software modules are as follows:

|                         |                                                                                                |
|-------------------------|------------------------------------------------------------------------------------------------|
| <b>pl7888_chkmark:</b>  | Checks for expiration of a previously set time mark.                                           |
| <b>pl7888_getdate:</b>  | Gets the current date.                                                                         |
| <b>pl7888_gettime:</b>  | Gets the current time.                                                                         |
| <b>pl7888_getwatch:</b> | Gets the date, time, and weekday in one call.                                                  |
| <b>pl7888_init:</b>     | Sets the current date and time on the clock.<br>Resets all time marks.                         |
| <b>pl7888_setalarm:</b> | Sets internal alarm marks to enable software to wait until a specific day and time.            |
| <b>pl7888_setmark:</b>  | Sets an internal future time mark for a specified time interval from the current day and time. |
| <b>pl7888_test:</b>     | Performs a self-test on the clock by reading the status bit on the clock.                      |
| <b>pl7888_wait:</b>     | Waits for a specified period of time.                                                          |
| <b>pl7888_weekday:</b>  | Gets the current day of the week.                                                              |

The clock maintains the current year in the form 0-99. For convenience, the year as specified (and returned) by all routines is in the form 1980-2079. In addition, the clock maintains the current hour in 24-hour mode (0-23). This is reported to the caller in this fashion.

This clock supports time measurement with resolution of one second. For the `pl7888_wait` routine, time intervals shorter than the minimum measurement will be measured using software program loops.

The `pl7888_setalarm` and `pl7888_setmark` (set interval) routines share the same internal structure. If the user attempts to set an interval mark using the same mark number as a previously set, unexpired alarm mark, the old mark will be overwritten by the new one. Marks may be reset at any time via calls to `pl7888_setmark` or `pl7888_setalarm`. It is the user's responsibility not to overwrite an existing mark before it has expired.

The watch chip on the 7888 CPU board will only resolve times down to one second. However, a few subroutine linkages feature times of hundredths of a second. This has been done to make the 7863 CPU board support and the 7888 CPU board support as similar as possible.

---

|                   |                                                                                                                                                                                                             |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7888_chkmark();<br/>int marknum;<br/>pl7888_chkmark(marknum);</pre>                                                                                                                       |
| Description:      | This routine checks for expiration of a previously set time mark. It verifies that the mark is set, then compares it with the current time. If the time is greater than the mark, the mark will be cleared. |
| Input Parameters: | <b>marknum</b> is an integer value specifying the time mark to be checked (0-9).                                                                                                                            |
| Returned Value:   | <b>pl7888_chkmark</b> contains the returned integer error code:<br><br>0 = interval has not expired<br>1 = interval has expired<br>-1 = invalid mark number<br>-2 = mark not set                            |
| Example:          | The following C program segment could be written to check mark number 2 to see if it has expired.<br><br><pre>errcode=pl7888_chkmark(2);</pre>                                                              |

|                      |                                                                                                                                                                                       |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:           | <pre>extern int pl7888_getdate();<br/>int year;<br/>int month;<br/>int day;<br/>year=pl7888_getdate(&amp;month,&amp;day);</pre>                                                       |
| Description:         | This routine gets the current date.                                                                                                                                                   |
| Input<br>Parameters: | <p><b>&amp;month</b> is a pointer to an integer where the value of month will be stored.</p> <p><b>&amp;day</b> is a pointer to an integer where the value of day will be stored.</p> |
| Returned<br>Value:   | <p><b>pl7888_getdate</b> contains the returned current year value (1980-2079).</p> <p><b>month</b> is filled (1-12).</p> <p><b>day</b> is filled (1-31).</p>                          |
| Example:             | <p>The following C program segment could be written to get the current date.</p> <pre>year=pl7888_getdate(&amp;month,&amp;day);</pre>                                                 |

---

|                   |                                                                                                                                                                                                                     |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7888_gettime();<br/>int hours;<br/>int minutes;<br/>int hundredths;<br/>hours=pl7888_gettime(&amp;minutes,<br/>&amp;hundredths);</pre>                                                            |
| Description:      | This routine gets the current time. Due to linkage limitations, the returned time will have a "hundredths of a second" field, rather than a "seconds" field.                                                        |
| Input Parameters: | <p><b>&amp;minutes</b> is a pointer to an integer where the value of minutes will be stored.</p> <p><b>&amp;hundredths</b> is a pointer to an integer where the value of hundredths of a second will be stored.</p> |
| Returned Value:   | <p><b>pl7888_gettime</b> contains the returned current hours (0-23).</p> <p><b>minutes</b> is filled (0-59).</p> <p><b>hundredths</b> is filled (0-5999).</p>                                                       |
| Example:          | <p>The following C program segment gets the current time.</p> <pre>hours=pl7888_gettime(&amp;minutes,<br/>&amp;hundredths);</pre>                                                                                   |

|                   |                                                                                                                                                                            |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7888_getwatch();<br/>int watcharray[8];<br/>pl7888_getwatch(watcharray);</pre>                                                                           |
| Description:      | This routine is used to obtain the date, time, and weekday in one call. Since the data cannot easily be passed, the caller should pass the address of an eight-word array. |
| Input Parameters: | <b>watcharray</b> is a pointer to an eight-word array where the time data will be stored.                                                                                  |
| Returned Value:   | <b>pl7888_getwatch</b> contains the returned integer error code:                                                                                                           |

0 = no error  
-1 = pointer is illegal (all zeros)

The watcharray contains the following information in order, if no error:

watcharray[0] = year (1980-2079)  
watcharray[1] = month (1-12)  
watcharray[2] = day (1-31)  
watcharray[3] = hours (0-23)  
watcharray[4] = minutes (0-59)  
watcharray[5] = seconds (0-59)  
watcharray[6] = hundredths of seconds (0-99)  
watcharray[7] = weekday (1 = Sunday, 2 = Monday, etc.)

**Example:** The following C program segment could be written to read all the time data.

```
int watcharray [8];
errcode=pl7888_getwatch (watcharray);
```

Structure:           extern int pl7888\_init();  
                      int year;  
                      int month;  
                      int day;  
                      int hours;  
                      int minutes;  
                      int seconds;  
                      int hundredths;  
                      pl7888\_init(year,month,day,hours,  
                      minutes,seconds, hundredths);

Description:       This routine sets the current date and time on the clock and clears any time marks that have been set. It also checks the clock with a call to the pl7888\_test routine. Note that this routine will change the time and date known to the STD-DOS system when the STD-DOS system is subsequently reset.

Input  
Parameters:       year is an integer value specifying the current year (1980-2079).

                  month is an integer value specifying the current month (1-12).

                  day is an integer value specifying the current day (1-31).

                  hours is an integer value specifying the current hours (0-23).

                  minutes is an integer value specifying the current minutes (0-59).

                  seconds is an integer value specifying the current seconds (0-59).

## pl7888\_init

hundredths is an integer value specifying the current hundredths of a second (0-99).

Returned  
Value:

pl7888\_init contains the returned integer error code:

- 0 = no error
- 1 = clock is not operational
- 2 = invalid date
- 3 = invalid time

Example:

The following C program segment could be written to set the current time to 07:00:00, September 1, 1986. The 7888 library software is also initialized.

```
errcode=pl7888_init(1986,9,1,7,0,0,0);
```

Structure:           extern int pl7888\_\_setalarm();  
                      int marknum;  
                      int weekday;  
                      int hours;  
                      int minutes;  
                      int seconds;  
                      pl7888\_\_setalarm(marknum,weekday,hours,  
                      minutes,seconds);

Description:           This routine sets an alarm mark for the  
                      specified time. pl7888\_\_chkmark can be used to  
                      determine if the alarm has occurred. Up to 10  
                      time marks may be set at a time. The ten time  
                      marks are set by both this routine and the  
                      pl7888\_\_setmark routine. If the specified time is  
                      earlier than the current time, the alarm will be  
                      set for the next coming week.

Input  
Parameters:           **marknum** is an integer value specifying the time  
                      mark to be set (0-9).

**weekday** is an integer value indicating the day of  
the week (1-7) of the alarm time.

**hours** is an integer value indicating the hour (0-  
23) of the alarm time.

**minutes** is an integer value indicating the  
minutes (0-59) of the alarm time.

**seconds** is an integer value indicating the  
seconds (0-59) of the alarm time.

## pl7888\_setalarm

Returned Value: pl7888\_setalarm contains the returned integer error code:

- 0 = no error
- 1 = invalid mark number
- 2 = invalid interval

Example: The following C program segment could be written to set mark number 2 for Monday, 7 a.m.

```
errcode=pl7888_setalarm(2,2,7,0,0);
```

---

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7888__setmark(); int marknum; int days; int hours; int minutes; int seconds; pl7888__setmark(marknum, days, hours, minutes,seconds);</pre>                                                                                                                                                                                                                                                                                                                                             |
| Description:      | <p>This routine sets a time mark, measured from the current time, whose expiration can be checked via calls to <code>pl7888__chkmark</code>. Up to 10 time marks may be set at a time. Note that the same 10 time marks may be set via calls to <code>pl7888__setalarm</code>. The maximum time for a mark interval is one week.</p>                                                                                                                                                                     |
| Input Parameters: | <p><b>marknum</b> is an integer value specifying the time mark to be set (0-9).</p> <p><b>days</b> is an integer value indicating the number of days (0-6) in the time interval.</p> <p><b>hours</b> is an integer value specifying the number of hours (0-23) in the time interval.</p> <p><b>minutes</b> is an integer value indicating the number of minutes (0-59) in the time interval.</p> <p><b>seconds</b> is an integer value indicating the number of seconds (0-59) in the time interval.</p> |
| Returned Value:   | <p><b>pl7888__setmark</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid mark number</li><li>-2 = invalid time interval</li></ul>                                                                                                                                                                                                                                                                                                 |

## pl7888\_\_setmark

Example:

The following C program segment could be written to set mark number 3 for one minute from now.

```
errcode=pl7888__setmark(3,0,0,1,0);
```

7888

C library

Structure:            `extern int pl7888_test();`  
                      `pl7888_test();`

Description:            This routine determines if the clock is operational by reading the clock status.

Input  
Parameters:            None

Returned  
Value:                `pl7888_test` contains the returned integer error code:

                      0 = no error

                      -1 = clock is not operational

Example:              The following C program segment tests the clock.

`errcode=pl7888_test();`

|                   |                                                                                                                                                                                                                                                                                                                                 |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Structure:        | <pre>extern int pl7888_wait(); int seconds; int milli; int micro; pl7888_wait(seconds,milli,micro);</pre>                                                                                                                                                                                                                       |
| Description:      | This routine can be used to wait for a specified period of time. The maximum waiting time interval is 60 seconds.                                                                                                                                                                                                               |
| Input Parameters: | <p><b>seconds</b> is an integer value indicating the number of seconds (0-59) in the wait interval.</p> <p><b>milli</b> is an integer value indicating the number of milliseconds (0-999) in the wait interval.</p> <p><b>micro</b> is an integer value indicating the number of microseconds (0-999) in the wait interval.</p> |
| Returned Value:   | <p><b>pl7888_wait</b> contains the returned integer error code:</p> <ul style="list-style-type: none"><li>0 = no error</li><li>-1 = invalid interval</li></ul>                                                                                                                                                                  |
| Example:          | <p>The following C program segment could be written to wait for 2.5 seconds.</p> <pre>errcode=pl7888_wait(2,500,0);</pre>                                                                                                                                                                                                       |

Structure:           extern int pl7888\_weekday();  
                      int wkday;  
                      wkday=pl7888\_weekday();

Description:         This routine gets the current day of the week.

Input  
Parameters:         None

Returned  
Value:             pl7888\_weekday contains the returned current  
                     day of the week (1-7), 1 = Sunday, 2 = Monday,  
                     etc.

Example:            The following C program segment gets the  
                     current day of the week.

```
wkday=pl7888_weekday();
```





MP MASTER



MA34218